Masarykova univerzita
Fakulta informatiky

Nízkolatenční přenosy 360 stupňového videa

Diplomová práca

Martin Piatka

Brno, jeseň 2020

Masarykova univerzita
Fakulta informatiky

Nízkolatenční přenosy 360 stupňového videa

Diplomová práca

Martin Piatka

Brno, jeseň 2020

Na tomto mieste sa v tlačenej práci nachádza oficiálne podpísané zadanie práce a vyhlásenie autora školského diela.

Vyhlásenie

Vyhlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Martin Piatka

Vedúci práce: RNDr. Miloš Liška Ph.D.

Poďakovanie

Chcel by som poďakovať RNDr. Milošovi Liškovi, Ph.D. za trpezlivosť a pomoc počas vedenia tejto práce. Ďalej chcem poďakovať Mgr. Martinovi Pulcovi za rady a odbornú pomoc. Na záver ďakujem svojej rodine, priateľom a kolegom z Laboratória pokročilých sieťových technológií za podporu.

Zhrnutie

Diplomová práca sa zaoberá návrhom a implementáciou získavania 360° videa pomocou kamerovej súpravy a jeho zobrazením pomocou tradičných zobrazovacích metód, ako aj pomocou headsetov pre virtuálnu realitu. Skladanie videa je akcelerované pomocou platformy CUDA. Na komunikáciu s headsetom je využité rozhranie OpenXR. Súčasťou práce je aj meranie výkonu výslednej implementácie, ako aj oneskorenia, ktoré do systému vnáša spracovanie 360° videa.

Kľúčové slová

UltraGrid, 360° video, CUDA, OpenGL, OpenXR

Obsah

1 Úvod **1**

2 360 stupňové video a projekcie **3**
2.1 Používané pojmy 3
2.2 Rektilineárna projekcia 4
2.3 Projekcia rybie oko 5
2.4 Ekvidistantná valcová projekcia 5
2.5 Kubická mapa 5
2.6 Zhrnutie 6

3 Kamerová súprava **7**
3.1 Rozmiestnenie kamier 7
3.2 Výber kamier 8
3.3 Výber objektívu 8
3.4 Statív 11
3.5 Synchronizácia 12
3.6 Rozhranie medzi súpravou a počítačom 12
3.7 Zhrnutie 13

4 Návrh skladania obrazu **15**
4.1 Požiadavky 15
4.2 Prehľad riešení na skladanie panorámy 15
4.2.1 OpenCV 15
4.2.2 Surround 360 15
4.2.3 NVIDIA VRWorks 360 Video 16
4.2.4 Zhrnutie 16
4.3 Priebeh skladania 16
4.4 Transformácia 17
4.5 Korekcia skreslenia objektívu 18
4.6 Kalibrácia 19
4.7 Kombinácia 20
4.7.1 Jednoduchý lineárny blending 21
4.7.2 Viacpásmový blending 22
4.8 Zhrnutie 24

5 Implementácia skladania obrazu **25**

5.1 Paralelizácia pomocou grafickej karty 25
5.2 Verejné rozhranie knižnice 26
5.2.1 Štruktúra `Stitcher_params` 26
5.2.2 Štruktúra `Cam_params` 26
5.2.3 Načítanie parametrov zo súboru 27
5.2.4 Trieda `Stitcher` 27
5.2.5 Trieda `Cuda_stream` 28
5.2.6 Trieda `Image_cuda` 29
5.3 Prehľad interných štruktúr a tried 29
5.4 Inicializácia 30
5.4.1 Vyhľadávanie prekryvov 30
5.5 Projekcia kamery 31
5.5.1 Výpočet transformácie 31
5.5.2 Vzorkovanie obrazových bodov vstupného snímku 32
5.6 Kombinácia 33
5.6.1 Kopírovanie neprekrývajúcich sa častí 34
5.6.2 Výpočet plynulých prechodov 34
5.6.3 Jednoduchý lineárny blending 35
5.6.4 Viacpásmový blending 36
5.6.5 Implementácia Laplacovej pyramídy 37
5.7 Súbežnosť pamäťových prenosov a výpočtov 40
5.8 Integrácia do programu UltraGrid 41
5.8.1 Rozhranie modulu 42
5.8.2 Získavanie vstupných snímok 43
5.8.3 Konverzia formátu obrazových bodov 43
5.8.4 Synchronizácia zaznamenávajúcich vlákien . . . 44
5.9 Zhrnutie 44

6 Implementácia zobrazovania **47**
6.1 Princíp vykresľovania 47
6.2 Trieda `GlProgram` 48
6.2.1 Vertex a fragment shader 48
6.3 Trieda `Model` 48
6.3.1 Generovanie modelu gule 49
6.4 Trieda `Texture` 50
6.5 Konverzia formátu obrazových bodov 50
6.6 Trieda `Scene` 51
6.6.1 Perspektívna a rotačná matica 51

6.7 Vytvorenie okna a inicializácia OpenGL kontextu . . . 51
6.7.1 Inicializácia okna . . . 52
6.7.2 Inicializácia OpenGL funkčných ukazovateľou . 52
6.8 Implementácia zobrazovanieho modulu `pano_gl` . . . 53
6.8.1 Rozhranie zobrazovacích modulov . . . 53
6.8.2 Návrh modulu `pano_gl` . . . 54
6.8.3 Udalosti a ich spracovanie . . . 55
6.8.4 Obmedzenie prekresľovania . . . 56
6.9 Zhrnutie . . . 56

7 Implementácia zobrazovania pomocou 3D headsetu 57
7.1 Prehľad rozhraní pre komunikáciu s headsetom . . . 57
7.1.1 Rozhranie ovládača od výrobcu . . . 57
7.1.2 OpenVR . . . 58
7.1.3 OpenHMD . . . 58
7.1.4 OpenXR . . . 58
7.1.5 Zhrnutie . . . 59
7.2 Inicializácia . . . 59
7.2.1 OpenXR Inštancia . . . 59
7.2.2 OpenXR systém . . . 60
7.2.3 OpenXR sedenie . . . 60
7.2.4 OpenXR priestor . . . 61
7.2.5 OpenXR pohľady . . . 61
7.2.6 OpenXR swapchain . . . 62
7.2.7 Kompozičné vrstvy . . . 62
7.3 Vykresľovací cyklus . . . 63
7.3.1 Časovanie snímok . . . 63
7.3.2 Získanie rotácie pohľadu . . . 63
7.3.3 Kreslenie snímky . . . 64
7.4 Optimalizácie . . . 65
7.4.1 Pamäťové prenosy s využitím Pixel buffer objektov 65
7.4.2 Nahrávanie video snímok z oddeleného vlákna 67
7.5 Zhrnutie . . . 68

8 Meranie výkonu 69
8.1 Časovač na inštrumentáciu . . . 69
8.2 Použité počítače . . . 70
8.3 Meranie skladania obrazu . . . 71

8.3.1 Testovanie očakávaného použitia 71
8.4 Merania zobrazovacieho modulu `openxr_gl` 73
8.4.1 Meranie výkonu nahrávania videosnímok do textúry 73
8.4.2 Meranie výkonu a oneskorenie vykreslenia snímok 74
8.5 Merania zobrazovacieho modulu `pano_gl` 76
8.6 Zhrnutie 77

9 Záver **79**

Bibliografia **81**

A Elektronická príloha **87**

B Meranie výkonu a oneskorenia implementácie využívajúcej knižnicu VRWorks **89**

1 Úvod

Za účelom lepšieho zážitku z videa sú technológie jeho produkcie a zobrazovania neustále zdokonaľované. Na zachytenie jemnejších detailov bolo postupne zvyšované rozlíšenie videa, no postupne sa dostávame do bodu, keď jeho ďalšie zvyšovanie pri použití štandardne veľkých displejov sledovaných z obvyklej vzdialenosti už neprináša výrazné vylepšenia vo vnímanej kvalite[1]. Preto je možné očakávať, že ďalšie zdokonalenie bude dosiahnuté nejakým novým spôsobom.

Jedným takým spôsobom je použitie 360° videa. V spojení s headsetom pre virtuálnu realitu je možné dosiahnuť veľmi živý a pohlcujúci zážitok, ktorý v divákovi vyvoláva pocit prítomnosti v zobrazovanom priestore. Vďaka tomu je 360° video vhodné na rôzne virtuálne prehliadky, alebo na zobrazovanie obsahu, ktorý vyžaduje veľmi široký zorný uhol, ako napríklad orchester z pohľadu dirigenta.

Táto práca sa snaží spojiť výhody 360° videa a nízkolatenčného živého prenosu videa vo vysokej kvalite rozšírením nástroja UltraGrid[1] o moduly pre získavanie 360° videa pomocou kamerovej súpravy a pre jeho zobrazovanie pomocou headsetu pre virtuálnu realitu alebo štandardne do okna na monitore.

Nástroj UltraGrid je vyvíjaný združením CESNET v spolupráci s Fakultou Informatiky Masarykovej univerzity. Zameriava sa na vysokú kvalitu obrazu a čo najmenšie oneskorenie.

Prvá časť práce sa zaoberá návrhom a implementáciou skladania videa. Aby sa implementované skladanie videa dalo použiť aj v iných projektoch, bolo implementované ako nezávislá knižnica. Na implementáciu bol využitý programovací jazyk C++ a na akceleráciu výpočtov bola využitá platforma CUDA.

Druhá časť práce popisuje návrh a implementáciu zobrazovania 360° videa. Ako grafické rozhranie pre vykresľovanie bolo zvolené rozhranie OpenGL. V snahe podporovať čo najviac modelov headsetov pre virtuálnu realitu od rôznych výrobcov na čo najviac platformách bolo na komunikáciu s headsetom zvolené rozhranie OpenXR.

Po dokončení implementácie bol meraný jej výkon a oneskorenie, ktoré je do nástroja UltraGrid vnesené spracovaním 360° obrazu.

---

1. `http://www.ultragrid.cz/`

2 360 stupňové video a projekcie

Video, s ktorým sa bežne stretávame, obsahuje obrazové dáta len pre určitý úsek priestoru, ktorý sa nachádza pred kamerou použitou na jeho zaznamenanie. Typicky sa jedná o zorný uhol 48.6° [2]. Video, ktoré miesto toho uchováva obrazové dáta z celého priestoru ktorý obklopuje kameru, nazývame 360 stupňové video.

Spôsob reprezentácie úseku priestoru pred kamerou na dvojrozmernú obrazovú plochu ktorý je pre ľudí intuitívny, spočíva v tom, že táto plocha funguje ako „okno“, cez ktoré sa pozorovateľ pozerá do zobrazovaného priestoru. Tento intutívny spôsob však s rastúcim zorným uhlom postupne prestáva fungovať, pretože nie je možné aby plocha ktorá slúži ako „okno“ obklopovala pozorovateľa z viacerých strán. Pre pochopenie reprezentácie 360 stupňového videa je preto potrebné zoznámiť sa s rôznymi alternatívnymi projekciami obrazu.

V tejto kapitole sú opísané nielen projekcie ktorými sa riadia skutočné objektívy, ale aj projekcie abstraktné, ktoré majú vhodné vlastnosti pre 360 stupňové video.

2.1 Používané pojmy

Za účelom ľahšieho a presnejšieho popísania rôznych projekcií a algoritmov si najprv zadefinujeme niekoľko pojmov, ktoré budú v tejto práci používané:

Optická os
: Pomyselná os, ktorá prechádza stredom snímacej plochy kamery a geometrickým stredom šošovky. Udáva smer, ktorým je kamera namierená.

Uhol θ
: Uhol medzi lúčom svetla vchádzajúceho do objektívu a optickou osou tohto objektívu označíme ako uhol θ.

Uhly λ**,** φ
: Okolo pozorovateľa si predstavíme jednotkovú guľu, ktorej stred leží v pozícii pozorovateľa. Všetky lúče svetla, ktoré potom prechádzajú stredom tejto gule (a teda dopadajú aj na pozorovateľa)

môžeme jednoducho kolmo premietnuť na jej povrch. Na vyjadrenie pozície týchto bodov na povrchu gule môžeme použiť uhly λ a φ, ktoré sú obdoba zemepisnej dĺžky a šírky.

Zorný uhol
Zorný uhol (anglicky field of view) predstavuje uhol ktorý zvierajú spojnice okrajových bodov obrazu s optickým stredom systému. Môže sa uvádzať ako horizontálny, vertikálny ale aj diagonálny.

Ohnisková vzdialenosť
Vzdialenosť zadného uzlového bodu objektívu od obrazovej plochy (optického snímača u digitálnych kamier). Spolu s rozmermi obrazovej plochy udáva zorný uhol. Keďže v tejto práci budeme pracovať s digitálnym obrazom, má zmysel na meranie tejto veličiny použiť ako jednotku počet pixelov.

$$f \text{ v pixeloch} = f \text{ v mm} \times \frac{\text{šírka obrazu v pixeloch}}{\text{šírka snímača v mm}} \tag{2.1}$$

Radiálna pozícia
Radiálna pozíciu označíme ako R. Táto veličina predstavuje vzdialenosť bodu od stredu obrazu.

2.2 Rektilineárna projekcia

S touto projekciou sa stretávame najčastejšie - je to projekcia ktorou sa riadi väčšina klasických objektívov. Táto projekcia zachováva rovnosť rovných čiar a preto je pre ľudí veľmi intuitívna. Rovnica tejto projekcie je

$$R = f \times \tan\theta \tag{2.2}$$

Nevýhodou tejto projekcie je, že sa dá použiť len pre uhly $\theta < 90°$. S rastúcim uhlom totiž dochádza k „naťahovaniu“ obrazu a pri uhle $90°$ dôjde k „natiahnutiu“ až do nekonečna.

2.3 Projekcia rybie oko

Rybie oko (anglicky fisheye) je trieda projekcií, ktorými sa riadia objektívy s veľmi veľkým zorným uhlom. Táto projekcia je použiteľná aj pre uhly $\theta >= 90°$.

Existuje viac typov tejto projekcie, no v tejto práci sa bude pracovať s lineárne škálovanou projekciou rybieho oka. Rovnica toho typu projekcie je

$$R = f \times \theta \tag{2.3}$$

Tento typ sa anglicky nazýva equidistant fisheye, pretože zachováva uhlové vzdialenosti.

2.4 Ekvidistantná valcová projekcia

Ekvidistantná valcová projekcia je projekcia, ktorou sa neriadi žiaden objektív, ale vďaka jej vlastnostiam je vhodná na reprezentáciu panoramatického obrazu. Táto projekcia sa niekedy nazýva aj geografická, pretože sa často používa na zobrazenie svetovej mapy.

Rovnice tejto projekcie sú

$$\begin{aligned} x &= (\frac{\lambda}{\pi} + 1) \times \frac{w}{2} \\ y &= (\frac{2\varphi}{\pi} + 1) \times \frac{h}{2} \end{aligned} \tag{2.4}$$

kde w, h odpovedajú rozmerom obrazu a λ, φ odpovedajú uhlom v horizontálnom a vertikálnom smere (obdoba zemepisnej dĺžky a šírky).

Vďaka jednoduchosti vzťahu medzi pozíciou obrazového bodu a uhlom pohľadu sa s touto projekciou veľmi dobre pracuje. Možno práve vďaka tomu patrí táto projekcia medzi najpoužívanejšie[3].

2.5 Kubická mapa

Kubická mapa (anglicky cube map) je zložená zo šiestich obrazov, ktoré tvoria steny kocky. Obraz každej zo stien je vytvorený pomocou

rektilineárnej projekcie s vertikálnym aj horizontálnym zorným uhlom 90°.

Táto forma reprezentácie panorámy je často používaná v počítačových hrách na vykresľovanie oblohy a rôzne grafické rozhrania ponúkajú jednoduchý spôsob ako takúto panorámu vykresľovať.

2.6 Zhrnutie

V tejto kapitole bolo definované čo sa myslí pod označením 360 stupňové video a bolo zavedených niekoľko pojmov, ktoré budú využívané aj v nasledujúcich kapitolách. Ďalej boli popísané rôzne projekcie obrazov spolu s ich rovnicami, ktoré budú neskôr využité pri implementácii skladania panorámy.

3 Kamerová súprava

Táto kapitola sa zaoberá hardvérovou stránkou získavania videa. Bude opísaná konštrukcia kamerovej súpravy a jednotlivé komponenty z ktorých sa skladá. Okrem výberu samotných kamier, objektívov a statívu bude tiež popísané aj rozhranie medzi kamerovou súpravou a spôsob synchronizácie jednotlivých kamier medzi sebou.

3.1 Rozmiestnenie kamier

Aby bolo možné správne vybrať vhodné kamery a objektívy je nutné najprv uvažovať o počte a umiestnení kamier v kamerovej súprave.

Na rozhraní obrazov zo susedných kamier, môžu kvôli nepresnostiam a paralaxnému javu vznikať rôzne artefakty. Vyšší počet kamier, umožňuje zaznamenávanie s väčšími prekrytiami, čo môže zabezpečiť väčšiu flexibilitu umiestnenia rozhraní medzi obrazmi z kamier. Rozhrania tak môžu byť posúvané na menej dôležité miesta v obraze, kde prípadné artefakty na rozhraniach pôsobia menej rušivo. Na druhú stranu s narastajúcim počtom kamier narastá počet týchto rozhraní ako aj potrebný výpočtový výkon.

Medzi populárne rozloženia kamier patrí uloženie kamier do prstenca - všetky kamery sú umiestnené vodorovne a medzi susedné kamery zvierajú spolu so stredom súpravy uhol $\frac{360^\circ}{n}$, kde n je celkový počet kamier. Toto rozloženie vyžaduje pomerne veľký zorný uhol vo vertikálnom smere. V niektorých prípadoch môže byť súprava doplnená o jednu kameru naviac, ktorá je namierená kolmo na ostatné kamery.

Ďalšie rozloženie je rozmiestniť kamery rovnomerne aj vo vertikálnom smere - súprava má tvar gule. Toto rozloženie je vhodné pri skladaní stereoskopických panorám, ktoré potrebujú prekrytie aspoň dvoch kamier v každom bode, pretože umožňuje použitie väčšieho počtu kamier.

Táto práca sa zaoberá skladaním monoskopických panorám, takže prekrytia kamier v každom bode nie sú nutné. Vzhľadom k potrebe video skladať v reálnom čase a možnostiam dostupnej nahrávacej karty bolo pre túto prácu vybrané prstencové rozloženie 4 kamier.

3.2 Výber kamier

Na účel skladania panorámy je vhodné aby mali kamery malý rozmer, aby bolo možné kamery upevniť na statív do čo najmenšej vzájomnej vzdialenosti. Toto je žiadúce, pretože s narastajúcou vzdialenosťou narastá aj vplyv paralaxného efektu, ktorý je popísaný v sekcii 3.4.

Ďalšou potrebnou vlastnosťou je možnosť použiť širokouhlý objektív typu rybie oko, ktorý dovolí zaznamenávať obraz s dostatočne veľkým zorným uhlom tak, aby bolo možné zachytiť celý priestor okolo kamerovej súpravy použitím primeraného počtu kamier.

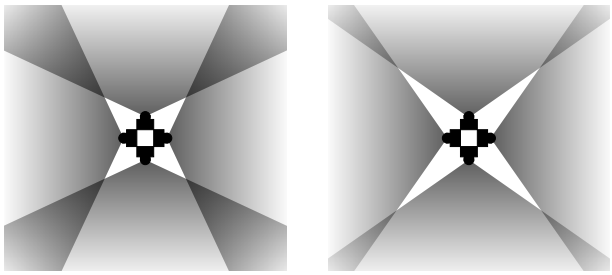
Na konštrukciu súpravy bola zvolená kamera Blackmagic Micro Studio Camera 4K[1]. Táto kamera aj napriek malým rozmerom ponúka záznam videa v UHD (3840×2160) rozlíšení s latenciou nižšou ako perióda snímok[4]. Na upevnenie objektívu používa systém Mikro 4/3, čo umožňuje výber z veľkého množstva kompatibilných objektívov. Veľkou výhodou je možnosť pripojenia externých synchronizačných hodín, ktoré umožnia jednotlivé kamery synchronizovať.

3.3 Výber objektívu

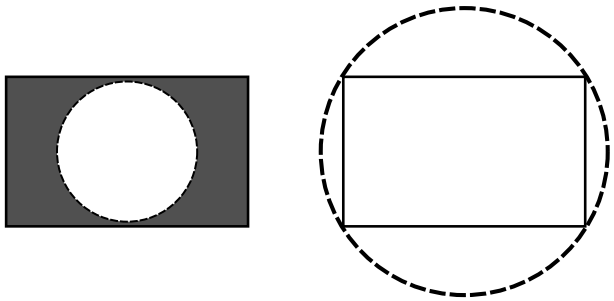
Pri výbere objektívu je dôležité zohľadniť jeho typ projekcie a zorný uhol, ktorý musí byť dostatočne veľký. V prípade usporiadania kamier do horizontálneho prstenca sa vyžaduje uhol väčší ako $\frac{360^\circ}{n}$ kde n je počet kamier. Okrem veľkosti oblasti prekrytia má zorný uhol vplyv aj na minimálnu vzdialenosť od súpravy v ktorej prekrytia vznikajú. Toto je znázornené na obrázku 3.1.

Objektívy typu rybie oko sa dajú rozdeliť do dvoch kategórií: Cirkulárne rybie oko a full-frame. Spôsob projekcie je pri obidvoch kategóriách rovnaký, líši sa len tým ktorá časť obrazu dopadá na obrazový snímač. Pri cirkulárnych objektívoch je snímaná okrúhly výsek obrazu pričom rohy obrazu sú nevyužité. Full frame objektívy premietajú obraz tak, že je celý obrazový snímač pokrytý, ale to má za následok orezanie zorného uhlu v niektorých smeroch. Porovnanie je možné vidieť na obrázku 3.2.

1. `https://www.blackmagicdesign.com/products/blackmagicmicrostudiocamera4k`



Obr. 3.1: Vplyv zorného uhla na veľkosť a pozíciu prekryvu obrazov



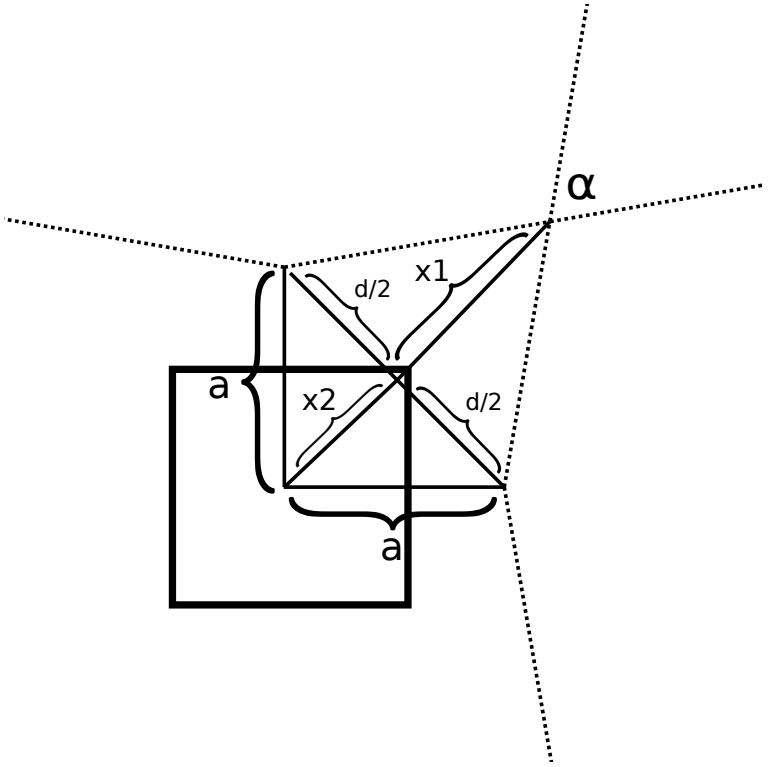
Obr. 3.2: Cirkulárne rybie oko (v ľavo) a full-frame rybie oko(v pravo)

Na vedomie je potrebné vziať fakt, že táto klasifikácia sa vzťahuje na použitie objektívu so snímačom, ktorého veľkosť odpovedá štandardu upevnenia objektívu. V prípade štandardu Micro 4/3 je tento rozmer $17.3mm \times 13.0mm$. Zvolená kamera však obsahuje snímač menší - $13.056mm \times 7.344mm$. Časť obrazu bude teda orezaná aj pri použití cirkulárneho objektívu.

Do kamerovej súpravy bol zvolený objektív Laowa 4mm f/2.8 Fisheye[2]. Jedná sa o cirkulárny objektív so zorným uhlom 210°. Pri portrétovej orientácii kamier (dlhšia hrana obrazu je zvislá) je zachytený celý zorný uhol 210°. Zorný uhol v horizontálnom smere je pri predpoklade ideálnej projekcie možné vypočítať vynásobením plného zorného uhlu pomerom veľkosti snímača ku priemeru plného kruhu: $\frac{7.344}{13} \times 210° = 118.63°$. Po kalibrácii je však táto hodnota kvôli skresleniam opísaných v kapitole 4.5 okolo 104.4°. Pri predpoklade

2. `https://www.venuslens.net/product/laowa-4mm-f-2-8-mft/`

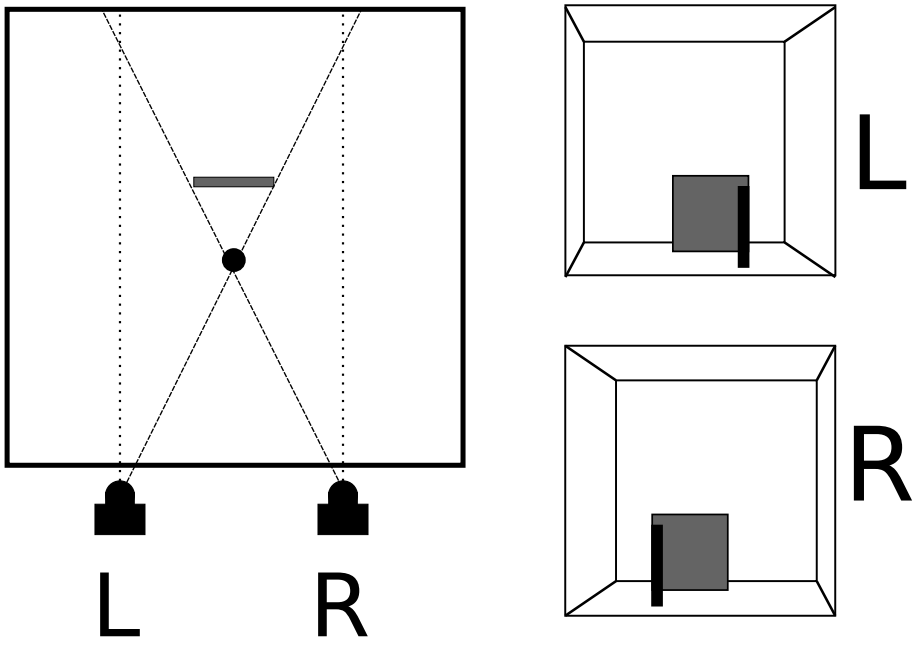
perfektne kolmého umiestnenia kamier je dosiahnutý prekryv s veľkosťou $104.4^\circ - 90^\circ = 14.4^\circ$. Na výpočet vzdialenosti bodu s najmenšou vzdialenosťou v ktorom dochádza k prekryvu tak ako bolo znázornené na obrázku 3.1 je potrebná vzdialenosť medzi optickými uzlovými bodmi kamier. Po odhadnutí tejto vzdialenosti pomocou rozmerov kamier a statívu je možné vypočítať, že k prekryvu začína dochádzať vo vzdialenosti približne $70cm$ od stredu statívu. Výpočet je znázornený na obrázku 3.3.



Obr. 3.3: Výpočet vzdialenosti prekryvu. Rozmer a udáva vzdialenosť optického uzlového bodu kamery od stredu súpravy. Rozmery $x2$ a d je možné vypočítať aplikáciou Pytagorovej vety. Vzdialenosť $x1$ je potom možné vypočítať ako $\frac{d/2}{tan(\alpha/2)}$, kde α je veľkosť prekryvu.

3.4 Statív

Okrem samozrejmých požiadavkou na statív ako možnosť upevnenia potrebného počtu kamier v požadovanom rozložení, je dôležité aby mohli byť kamery upevnené čo najbližšie k sebe. S narastajúcou vzdialenosťou medzi kamery totiž narastá účinok paralaxného javu (ukážka na obrázku 3.4).



Obr. 3.4: Ukážka paralaxného javu. Z pohľadu ľavej kamery je tmavý stĺp napravo od šedej plochy, zatiaľ čo z pohľadu pravej kamery sa javí byť naľavo.

Pre túto prácu bol vybraný statív 360RIZE 360Helios 7[3]. Tento statív je navrhnutý priamo pre použité kamery a preto ponúka najmenšiu vzdialenosť medzi kamerami ako je možné, pri použití štyroch vybraných kamier v prstencovej konfigurácii. Pri využití väčšieho počtu kamier by bolo možné dosiahnuť aj nižšie vzdialenosti, ale za cenu nevýhod popísaných v sekcii 3.1.

3. https://shop.360rize.com/product/360rize-360helios-6-360-plug-n-play-rig-for-blackmagic/

3.5 Synchronizácia

V prípade ak jednotlivé kamery v kamerovom systéme nie sú synchronizované, môžu snímať jednotlivé snímky videa v jemne odlišných okamžikoch. Tieto časové rozdiely sú však dostatočne veľké na to, aby pri pohybujúcich sa objektoch cez rozhrania medzi kamerami vznikali artefakty. Tento efekt je obzvlášť viditeľný pri pohybe celou kamerovou súpravou. Jednotlivé kamery je preto vhodné medzi sebou synchronizovať.

Vybrané kamery podporujú pripojenie externého generátoru synchronizačného signálu. Tento generátor generuje periodický pulzný signál, ktorý sa v čase opakuje v závislosti od zvolenej snímkovej frekvencie. Pripojenie všetkých kamier v súprave ku jednému generátoru synchronizačného signálu zaistí, že budú snímky zhotovovať v rovnakom momente. V kamerovej súprave bol použitý generátor Blackmagic Mini Converter Sync Generator[4], ktorý ponúka dostatočný počet výstupov.

3.6 Rozhranie medzi súpravou a počítačom

Na prenos obrazových dát medzi kamerami a počítačom je použitý štandard SDI (Serial Digital Interface). Tento štandard používa na prenos dát koaxiálny kábel a jeho použitie je pomerne rozšírené v profesionálnej video technike.

Na vstup videa do počítača bola použitá nahrávacia karta Decklink 8K Pro[5]. Táto karta disponuje štyrmi SDI vstupmi z ktorých je možné nahrávať video súčasne. Vstup videa v rozlíšení 8k (7680×4320) je podporovaný takým spôsobom, že každá snímka je rozdelený na štyri časti, pričom sa každá časť prenáša oddelenými koaxiálnymi káblami, ktorých signál je vzájomne synchronizovaný. Keďže kamery sú navzájom synchronizované tak ako bolo opísané v sekcii 3.5, je možné získať snímky zo všetkých štyroch kamier naraz ako jednu

4. https://www.blackmagicdesign.com/products/miniconverters/techspecs/W-CONM-15
5. https://www.blackmagicdesign.com/products/decklink/techspecs/W-DLK-34

snímku v 8k rozlíšení. Toto zaistí korektnú sychronizáciu vstupných dát na strane počítača.

3.7 Zhrnutie

Táto kapitola obsahuje popis jednotlivých hardvérových komponentov potrebných na zaznamenávanie obrazových dát potrebných na vytvorenie panoramatického videa.

Zvolené boli tieto komponenty:

Kamery : 4x Blackmagic Micro Studio Camera 4K

Objektív : Laowa 4mm f/2.8 Fisheye

Statív : 360RIZE 360Helios

Vstup dát do PC : Decklink 8K Pro

Táto kamerová súprava umožňuje získavať 4 synchronizované video prúdy, každý v rozlíšení 4K a snímkovou frekvenciou 30 snímkov za sekundu. Spoločne tieto obrazové prúdy pokrývajú celý priestor okolo súpravy, pričom medzi susednými kamerami vznikajú prekrytia s veľkosťou okolo 14.4°.

4 Návrh skladania obrazu

4.1 Požiadavky

Cieľom práce je vytvoriť modul programu UltraGrid, ktorý je schopný skladať video panorámu v reálnom čase. Medzi výhody tohto programu patrí schopnosť prenášania videa vo vysokých rozlíšeniach s minimálnym oneskorením, preto sa aj od modulu na skladanie panorámy očakáva, že jeho výkon bude postačovať na video v rozlíšení 8K so snímkovou frekvenciou aspoň 30 snímkov za sekundu, bez výrazného zavedenia oneskorenia naviac.

Vzhľadom k navrhnutej kamerovej súprave sa vyžaduje podpora prstencovej konfigurácie kamier a objektívov typu rybie oko. Očakávaným výstupom je monoskopická panoráma s ekvidistantnou valcovou projekciou.

4.2 Prehľad riešení na skladanie panorámy

4.2.1 OpenCV

Knižnica OpenCV[1] ponúka funkcie na skladanie panorám[5], ale ich požitie je skôr mierené na skladanie statických snímkov a aj napriek akcelerácii niektorých častí výpočtu pomocou grafickej karty sa čas skladania jedného snímku pohyboval okolo sekundy, čo na skladanie videa v reálnom čase nie je dostatočné.

4.2.2 Surround 360

V roku 2016 firma Facebook zverejnila pod voľnou licenciou systém Surround 360[6]. Tento projekt obsahuje okrem softvéru aj schémy hardvéru kamerovej súpravy. Surround 360 sa zameriava na kvalitné skladanie stereoskopickej panorámy, a preto namiesto skladania panorámy v reálnom čase počas nahrávania sú najprv zaznamenávané obrazové dáta z každej kamery a samotné skladanie prebieha až po dokončení natáčania.

---

1. `https://opencv.org/`

4.2.3 NVIDIA VRWorks 360 Video

VRWorks 360 Video je knižnica vyvíjaná firmou NVIDIA. Podporuje skladanie stereoskopických ako aj monoskopických panorám[7]. Na dosiahnutie potrebného výkonu na skladanie týchto panorám v reálnom čase je výpočet presunutý na grafickú kartu pomocou platformy CUDA. Táto knižnica nemá otvorený zdrojový kód a bola k dispozícii ako balík obsahujúci hlavičkové súbory obsahujúce definície rozhrania a binárne súbory. V roku 2020 NVIDIA ohlásila ukončenie vývoja tejto knižnice a vývojový balík už nie je možné zo stránok tejto firmy stiahnuť. Pred ukončením vývoja bola pre skladanie zvolená táto knižnica. Porovnanie výkonu a oneskorenia prvotnej implementácie, ktorá využívala túto knižnicu, s výslednou implementáciou je v dodatku B.

4.2.4 Zhrnutie

Preskúmané riešenia buď nespĺňajú výkonnostné požiadavky na skladanie panorám vo vysokom rozlíšení v reálnom čase, alebo prestali byť dostupné. Z tohto dôvodu som sa preto rozhodol pre účel tejto práce implementovať vlastnú knižnicu.

4.3 Priebeh skladania

Táto sekcia obsahuje stručný popis fungovania navrhnutého skladača. Priebeh skladania panorámy je možné zhrnúť do nasledujúcich krokov:

1. Inicializácia skladača: Keďže sa dá predpokladať že kamery nebudú počas skladania meniť vzájomnú pozíciu, je možné pre kamerovú súpravu dopredu získať kalibrované informácie o vzájomnej pozícii kamier a použitých objektívoch. Tieto informácie sú skladaču poskytnuté pri inicializácii a sú použité pre každú snímku.

2. Transformácia do cieľovej projekcie: Vstupné snímky z kamerovej súpravy sú zo zdrojovej projekcie (rybie oko) transformované do projekcie cieľovej (ekvidistantná valcová).

3. Kombinácia: Výsledkom predchádzajúceho kroku je niekoľko obrazov obsahujúcich určitú oblasť výslednej panorámy. V tomto

kroku dochádza k ich kombinácii takým spôsobom aby nedochádzalo k výrazným hranám ktoré by pôsobili rušivo.

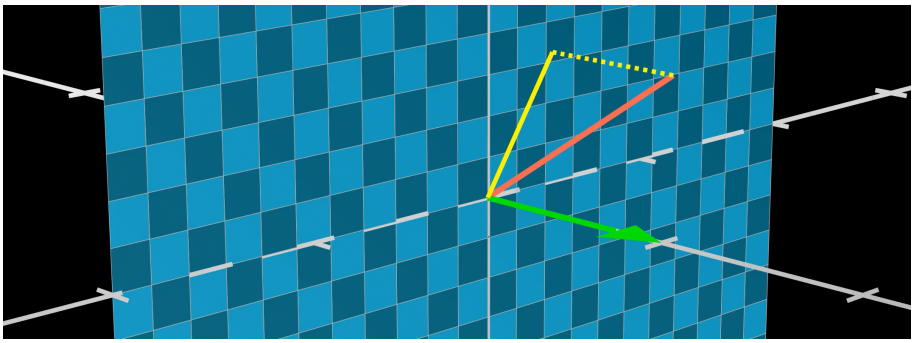
4.4 Transformácia

Proces transformácie zachováva hodnoty obrazových bodov ktoré prislúchajú rovnakej pozícii v priestore okolo pozorovateľa. Mení sa len vzťah medzi touto pozíciou a súradnicami obrazového bodu. Na transformácie medzi jednotlivými projekciami je možné využiť rovnice projekcií, ktoré boli opísané v kapitole 2. Tento výpočet vyžaduje pomerne presné informácie o orientácii kamery v priestore a použitom objektíve. Vyhľadávanie bodov v obraze s projekciou rybie oko, ktoré prislúchajú bodom v cieľovom obraze s ekvidistantnou valcovou projekciou prebieha v nasledujúcich krokoch:

1. Pre daný obrazový bod vo výslednom obraze sú pomocou rovníc 2.4 vypočítané uhly λ a φ. Následne je vypočítaný jednotkový vektor $\vec{v}$, ktorého vertikálny a horizontálny sklon odpovedá týmto uhlom.

2. Uhol ktorý medzi sebou zvierajú vektor $\vec{v}$ a optická os kamery z ktorej pochádza zdrojový obraz je podľa definície uhol θ. Jeho dosadením do rovnice 2.3 môžeme získať vzdialenosť r hľadaného bodu od stredu zdrojového obrazu.

3. Na určenie konkrétnych súradníc hľadaného bodu v zdrojovom obraze je okrem vzdialenosti r nutné zistiť aj smer v ktorom tento bod od stredu leží. Zdrojový obraz je možné si predstaviť ako plochu, ktorej orientácia je zhodná s orientáciou kamery (t.j. kolmá na optickú os kamery). Následne je na túto plochu vykonané kolmé premietnutie vektoru $\vec{v}$. Normalizáciou a následným vynásobením zložiek tohto vektoru vzdialenosťou r potom dostávame vertikálnu a horizontálnu vzdialenosť hľadaného bodu od stredu zdrojového obrazu. Tento krok výpočtu je znázornený na obrázku 4.1.



Obr. 4.1: Kolmé premietnutie vektoru $\vec{v}$. Vektor reprezentujúci optickú os kamery je znázornený zelenou farbou. Vektor $\vec{v}$ je znázornený červenou. Výsledný premietnutý vektor je znázornený plnou žltou čiarou.

4.5 Korekcia skreslenia objektívu

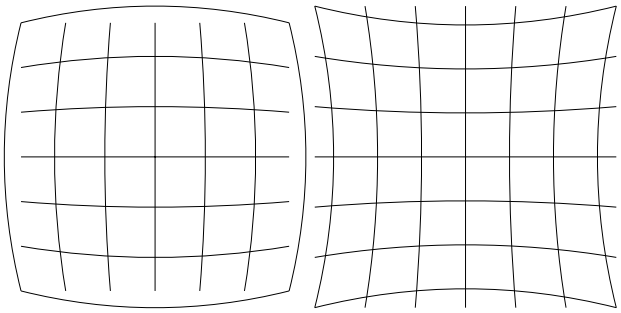
Projekcie spomenuté v kapitole 2 popisujú matematicky ideálne vlastnosti objektívov. Reálne objektívy sa však týmto vlastnostiam len približujú a dochádza ku rôznym skresleniam. Väčšinou sa jedná o takzvané radiálne skreslenie. Radiálne skreslenie je možné rozdeliť na pozitívne, negatívne, alebo prípadne ich kombináciu[8].

Pozitívne radiálne skreslenie - alebo aj pincushion distortion vzniká ak je bod premietnutý do väčšej vzdialenosti od optického stredu ako v ideálnom prípade.

Negatívne radiálne skreslenie - alebo aj barrel distortion vzniká ak je bod premietnutý do menšej vzdialenosti od optického stredu ako v ideálnom prípade.

Miera skreslenia je približne rovnaká vo všetkých bodoch s rovnakou vzdialenosťou od optického stredu. Táto vlastnosť nám umožňuje uvažovať o funkcii f na korekciu vypočítanej vzdialenosti od optického stredu podľa rovníc projekcií.

$$r_{corrected} = f(r_{ideal}) \tag{4.1}$$



Obr. 4.2: negatívne radiálne skreslenie (na ľavo) a pozitívne radiálne skreslenie (na pravo) (Zdroj obrázkov: `https://en.wikipedia.org/wiki/File:Barrel_distortion.svg` `https://en.wikipedia.org/wiki/File:Pincushion_distortion.svg`)

Vo svojej implementácii som použil rovnaký model korekcie radiálneho skreslenia ako zbierka programov The PanoTools[2]. Tento model je založený na polynóme tretieho stupňa

$$r_{corrected} = (ar_{ideal}^3 + br_{ideal}^2 + cr_{ideal} + d)r_{ideal} \tag{4.2}$$

kde a, b, c, d sú koeficienty skreslenia a d je definované ako $d = 1 - (a + b + c)$. Vzdialenosti r sú v tomto modele normalizované takým spôsobom, že vzdialenosť 1.0 odpovedá polomeru najväčšieho kruhu, ktorý sa zmestí do snímku celý[9].

4.6 Kalibrácia

Na zvýšenie presnosti informácií o orientácii kamier a použitých objektívoch je možné vykonať kalibráciu. Proces kalibrácie je možné rozdeliť na niekoľko krokov:

vyhľadávanie význačných bodov
: V obraze sa vyhľadávajú body, ktoré sú niečím výrazné. Typicky sa jedná o rohy.

2. `http://panotools.sourceforge.net/`

identifikácia
: V oblasti kde sa susedné snímky prekrývajú sa hľadá, ktoré význačné body si navzájom prislúchajú.

optimalizácia
: Parametre projekcie upravia takým spôsobom, aby vzdialenosť bodov ktoré si prislúchajú bola po projekcii minimálna.

Kroky vyhľadávania význačných bodov a identifikácia sa vykonávajú pomocou detektora a deskriptora význačných bodov. Medzi takéto detektory patrí napríklad ORB (Oriented FAST and Rotated BRIEF). Tento detektor najprv vyhľadáva význačné body pomocou algoritmu FAST (Features from accelerated segment test)[10], ktorý je založený na porovnávaní intenzity obrazového bodu a jeho okolia[11]. Po vyhľadaní význačných bodov v obrazoch zo susedných kamier sú pre tieto body vytvorené deskriptory pomocou algoritmu BRIEF (Binary Robust Independent Elementary Features). Na základe týchto deskriptorov sú v oblasti prekryvu nájdené páry význačných bodov, ktoré odpovedajú rovnakému bodu v reálnom svete.

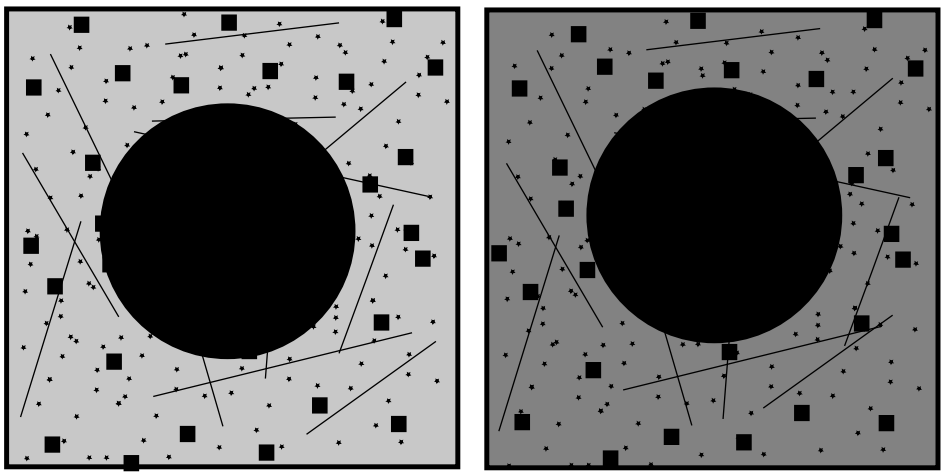
Získané dvojice význačných bodov, by sa mali v ideálnom prípade po transformácii do výslednej projekcie s presnými parametrami zobraziť do identických súradníc. Na základe približných parametrov projekcie, ktoré sú zadané užívateľom, je možné vypočítať súradnice, kam sa tieto body zobrazia. Vzdialenosť medzi výslednými bodmi z dvojice potom môžeme označiť ako chybu. Cieľom optimizácie je upraviť parametre projekcie tak, aby bola táto chyba pre všetky dvojice čo najmenšia. Optimalizáciu je možno vykonať ako minimalizáciu súčtu štvorcov chýb pomocou nejakej iteračnej metódy, ako napríklad algoritmus Levenberg–Marquardt, ktorý je použitý knižnicou libpano13[12].

Proces kalibrácie je výpočtovo náročný a preto pre skladanie v reálnom čase radšej predpokladáme, že jednotlivé kamery nebudú vzájomne voči sebe meniť orientáciu a kalibráciu vykonáme len raz.

4.7 Kombinácia

Po transformácií n vstupných snímkov z n kamier, potrebujeme tieto snímky nejakým spôsobom skombinovať do jedného výstupného. Naj-

jednoduchšia metóda je použitie binárnej masky, ktorá pre každý bod obsahuje informáciu, ktorý zo snímkov sa má použiť. Kvôli paralaxnému efektu spôsobeného vzájomným posunom jednotlivých kamier v súprave však nie je možné obrazy v prekrývajúcich sa oblastiach zarovnať perfektne. Táto metóda preto vo výstupnom obraze zanecháva ostré hrany ktoré pôsobia rušivo. Z toho dôvodu je žiadúce vytvoriť v prekrývajúcich oblastiach čo najplynulejší prechod medzi obrazmi zo susedných kamier. Proces výpočtu tohto prechodu nazývame blending. V tejto práci bude implementovaný jednoduchý lineárny blending a neskôr aj zložitejší viacpásmový blending. Algoritmus lineárneho blendingu bol vybraný kvôli jeho jednoduchusti a rýchlosti výpočtu, multiband blending zas ponúka dobrý kompromis medzi kvalitou a zložitosťou výpočtu[13]. Výsledky opisovaných techník blendingu budú znázornené pomocou testovacích obrázkov 4.3a a 4.3b.



(a) a (b) b

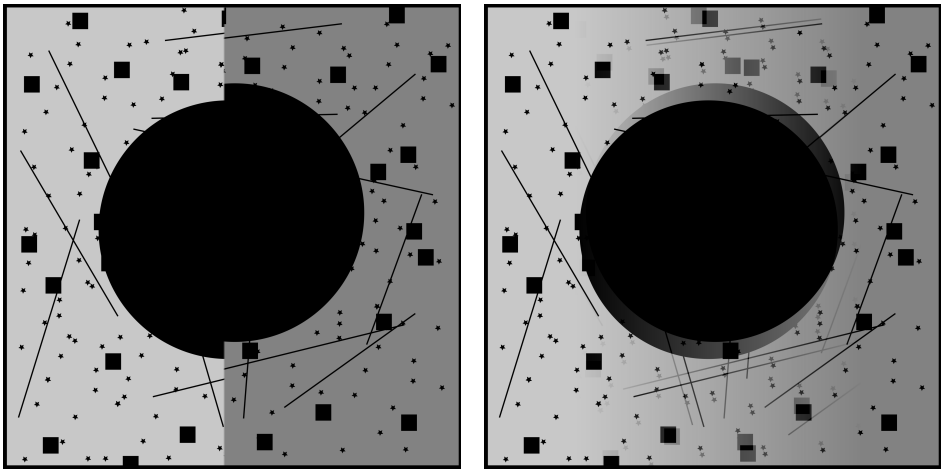
Obr. 4.3: Obrázky použité na znázornenie techník blendingu.

4.7.1 Jednoduchý lineárny blending

Lineárny blending spočíva v kombinácii hodnôt obrazových bodov dvoch obrazov, pomocou váh. Výpočet sa dá zapísať pomocou vzorca $pix_{dst} = w \times pix_{src1} + (1 - w) \times pix_{src2}$, kde hodnota w lineárne klesá

od 1 na začiatku prechodu po 0 na konci. Táto technika je veľmi jednoduchá na implementáciu a je nenáročná na výpočtový výkon, ale v niektorých situáciách je kvalita výsledku príliš závislá na voľbe šírky prechodu.

Vhodná veľkosť prechodu závisí od veľkosti od veľkostí výrazných čŕt vo vstupných obrazoch. V prípade ak je šírka prechodu príliš veľká môže dochádzať k „zdvojeniu“ niektorých čŕt (viz obr. 4.4b), naopak ak je šírka príliš malá môžu vo výslednom obraze vzniknúť rušivé hrany (viz obr. 4.4a). Z tohto dôvodu je táto metóda nevhodná pre obrazy ktoré obsahujú malé aj veľké črty zároveň. Výsledky tejto techniky sú znázornené na obrázkoch 4.4.



(a) feather malá šírka (b) feather veľká šírka

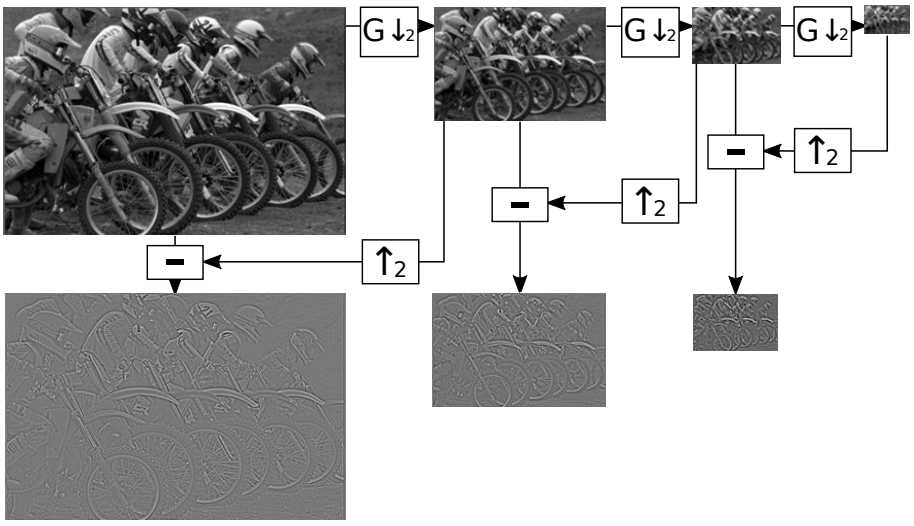
Obr. 4.4: Výsledky jednoduchého linueárneho blendingu s rôznymi šírkami.

4.7.2 Viacpásmový blending

Problémy so zvolením správnej šírky prechodu sa pokúša vyriešiť viacpásmový blending (anglicky Multi-band blending)[14]. Myšlienkou tejto metódy je, že pre malé črty sa použije malá šírka prechodu, zatiaľ čo veľké črty sú spracované s väčšou šírkou. Toto je umožnené faktom, že menšie črty vyžadujú väčšie zmeny hodnôt susedných obrazových bodov. Veľkosť zmeny hodnôt v pomere ku vzdialenosti

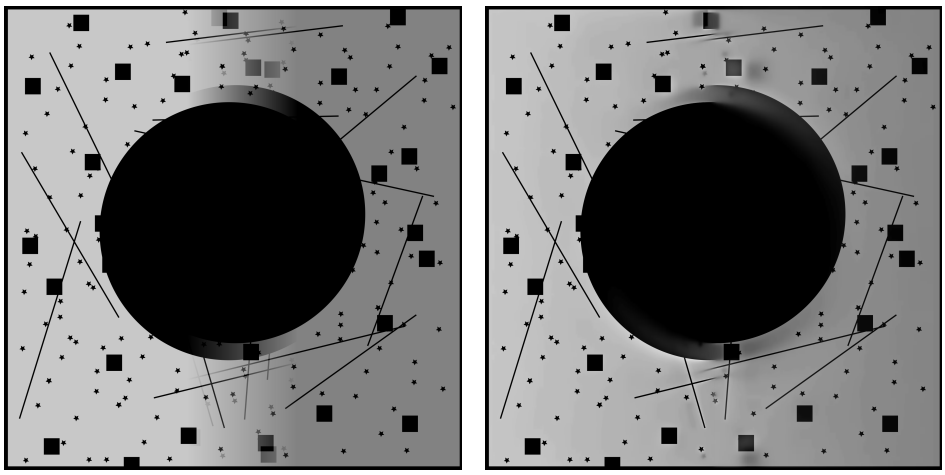
daných obrazových bodov označujeme ako priestorovú frekvenciu. Rozdelenie obrazu na jednotlivé frekvenčné zložky nám umožní ich spracovať oddelene.

Na získanie jednotlivých zložiek je použitá Laplacova pyramída. Idea tejto pyramídy spočíva v myšlienke, že obraz s nižším rozlíšením môže obsahovať len zložky s nižšou frekvenciu. Pyramída je zostavená z niekoľko poschodí, pričom každé poschodie je tvorené obrazom s polovičným rozlíšením ako v predchádzajúcom poschodí. Pri výpočte jednotlivých poschodí je od obrazu odčítaná jeho menej detailná varianta, ktorá vznikne jeho zmenšením a opätovným zväčšením. Každé poschodie, s výnimkou posledného, potom obsahuje len detaily ktoré boli stratené zmänšením obrazu pre následujúce poschodie. Keďže pre posledné poschodie už nemá následníka, miesto rozdielu obsahuje jednoducho len zmenšený obraz. Postupným zväčšovaním tohto obrazu a pripočítavaním rozdielov na jednotlivých poschodiach je potom možné bezstrátovo rekonštruovať pôvodný obraz v plnom rozlíšení. Postup pri konštrukcii pyramídy je znázornený na obrázku 4.5.



Obr. 4.5: Schéma konštrukcie pyramídy. Prekreslené z Perceptual Image Enhancement, Lark Kwon Choi, Todd Goodall, Deepti Ghadiyaram, Alan C. Bovik

Pri výpočte viacpásmového blendingu sú najprv vypočítané Laplacove pyramídy pre obidva vstupné obrazy. Následne je pre každé poschodie týchto pyramíd vypočítaný jednoduchý lineárny prechod. Výsledok je získaný rekonštrukciou obrazu z takto získanej pyramídy. Porovnanie výsledku tejto techinky s lineárnym blendingom je možné vidieť na obrázku 4.6.



(a) feather stredná šírka (b) multiband blending

Obr. 4.6: Porovnanie lineárneho blendingu so strednou šírkou a multiband blendingu.

4.8 Zhrnutie

V tejto kapitole bol popísaný návrh procesu skladania video panorámy a delí ho na niekoľko krokov. Ku každému kroku obsahuje krátky popis ako aj stručné vysvetlenie použitých algoritmov. Implementácia týchto algoritmov je detailnejšie opísaná v následujúcej kapitole.

5 Implementácia skladania obrazu

Táto kapitola sa zaoberá návrhom a implementáciou knižnice na skladanie 360° video snímok v reálnom čase. Ďalej bude opísaná aj integrácia tejto knižnice do programu UltraGrid.

5.1 Paralelizácia pomocou grafickej karty

Väčšina výpočtov popísaných v tejto kapitole prebieha pre výstupné obrazové body nezávisle od seba - na výpočet hodnoty obrazového bodu vo výslednom obraze nepotrebujeme vedieť hodnoty jeho susedných bodov. Postup výpočtu je navyše až na vstupné dáta pre všetky body tiež totožný. Výpočty s týmito vlastnosťami sú triviálne paralelizovateľné.

Moderné grafické karty ponúkajú možnosť spúšťania užívateľom definovaných výpočtov. Táto funkcionalita sa označuje ako GPGPU (General-purpose computing on graphics processing units). Súčasné modely grafických kariet obsahujú tisíce výpočtových jednotiek[15], ktoré umožňujú simultánny beh väčšieho množstva výpočtových vlákien ako umožňujú bežne dostupné procesory. Za účelom dosiahnutia dostatočného výkonu na skladanie panorámy v reálnom čase budú preto výpočty vykonávané zväčša na grafickej karte.

Ako GPGPU platformu som pre tento projekt zvolil platformu CUDA od firmy NVIDIA, pretože je táto platforma v programe UltraGrid už využívaná na rôzne účely ako kompresia videa a výpočet kontrolných a samoopravujúcich kódov. Táto voľba preto uľahčí integráciu do prekladového systému a výrazne zjednoduší niektoré optimalizácie.

Výpočty na grafickej karte môžu pracovať len s dátami uloženými v operačnej pamäti grafickej karty. Táto pamäť je oddelená od bežnej operačnej pamäte ktorá sa používa pri výpočtoch na klasických procesoroch. Na prenos dát medzi týmito pamäťami je nutné použiť špeciálne funkcie poskytnuté ovládačmi grafickej karty. Na rozlíšenie týchto typov pamätí budú v tejto práci použité pojmy GPU pamäť a CPU pamäť.

5.2 Verejné rozhranie knižnice

Pojem verejné rozhranie označuje súbor dátových štruktúr a funkcií, ktoré sú poskytnuté používateľom knižnice. Toto rozhranie by malo vystaviť všetku funkcionalitu a zároveň byť čo najjednoduchšie a najmenšie.

Hlavičkové súbory `.h` a `.hpp` ktoré toto rozhranie obsahujú sú od ostatných súborov oddelené do zložky `gpustitch/include`, zatiaľ čo ostatné súbory zdrojového kódu vrátane hlavičkových súborov, ktoré nie sú verejné sú uložené v zložke `gpustitch/src`. Toto rozdelenie okrem zvýšenia prehľadnosti pre užívateľov tejto knižnice zároveň uľahčuje napísanie inštalačného skriptu tak, aby boli inštalované len potrebné súbory.

Toto verejné rohranie bude opísané v nasledujúcich podsekciách.

5.2.1 Štruktúra `Stitcher_params`

Táto štruktúra obsahuje všeobecné nastavenia pre skladanie panorámy.

`width` a `height` - slúži na nastavenie šírky a výšky výslednej panorámy

`blend_algorithm` - slúži na výber algoritmu použitého na miešanie obrazu v oblastiach v ktorých sa prekrývajú obrazy zo susedných kamier

`feather_width` - slúži na nastavenie šírky prechodu ak je zvolený lineárny blending

`multiband_levels` - slúži na nastavenie počtu levelov pyramíd ak je zvolený multiband blending

5.2.2 Štruktúra `Cam_params`

Táto štruktúra obsahuje informácie o rotácii kamery a použitom objektíve. Tieto nastavenia sú pre každú kameru individuálne a použitie rôznych kamier a objektívov v jednej súprave je povolené.

`width` a `height` - obsahuje šírku a výšku vstupného obrazu z kamery

`focal_len` - Ohnisková vzdialenosť meraná v obrazových bodoch, tak ako bolo popísané v sekcii 2.1

`distortion[4]` - Pole obsahujúce koeficienty skreslenia objektívu, jedná sa o koeficienty a, b, c, d opísané v sekcii 4.5

`x_offset` **a** `y_offset` - V prípade ak optický stred objektívu neleží v strede obrazu, je pomocou týchto položiek možné definovať rozdiel v polohe týchto stredov

`yaw`, `pitch` **a** `roll` - Určuje rotáciu kamery v priestore

5.2.3 Načítanie parametrov zo súboru

Z praktického hľadiska je žiadúce aby bolo možné parametre kamier meniť rýchlo a jednoducho. V súbore `config_utils.hpp` je preto deklarovaná funkcia `read_params`, ktorá umožňuje prečítanie parametrov zo súboru.

Ako formát pre špecifikáciu týchto parametrov som zvolil textový formát TOML[1] (Tom's Obvious, Minimal Language). Tento formát umožňuje jednoduchý zápis vo forme kľúčov s hodnotami a obsahuje aj podporu komentárov. Vďaka týmto vlastnostiam je zrozumiteľný a editovateľný aj pre menej skúsených používateľov.

5.2.4 Trieda `Stitcher`

Táto trieda slúži ako rozhranie na obsluhu samotného skladania panorámy vrátane prenosov vstupných a výstupných dát. Za účelom oddelenia implementačných detailov od rozhrania je použitý známy vývojový vzor PImpl, ktorý spočíva vo vytvorení triedy, ktorá obsahuje ukazovateľ na skutočnú implementáciu na ktorú potom presmeruje všetky volania členských funkcií. Keďže jazyk C++ podporuje ukazovatele na dopredne deklarované typy, nemusí hlavičkový súbor takéhoto rozhrania vôbec obsahovať definície typov použitých v implementácii.

`Stitcher(Stitcher_params, const vector<Cam_params>)`
Konštruktor, ktorý vytvorí a inicializuje inštanciu skladača s danými parametrami.

1. `https://github.com/toml-lang/toml`

`submit_input_image()`
: Funkcia na predanie vstupného snímku do skladača. Parameter `cam_idx` určuje ktorej kamere predávaná snímka prislúcha. Parametre `w` a `h` udávajú rozmer vstupného snímku, `pitch` udáva počet bytov medzi začiatkami po sebe následujúcich riadkov obrazových bodov. Zdroj dát môže byť uložený v CPU ale aj GPU pamäti. Na rozlíšenie umiestnenia zdrojových dát slúži parameter `mem_kind`. Táto funkcia je dostupná aj v `_async` variante, ktorá neblokuje cpu a vráti riadenie pred dokončením operácie.

`stitch()`
: Volaním tejto funkcie je naplánované skladanie panorámy. Táto funkcia obsahuje synchronizáciu s asynchrónnymi vstupnými prenosmi a blokuje kým sa nedokončia. Samotné skladanie panorámy prebieha asynchrónne.

`download_stitched()`
: Táto funkcia slúži na získanie výslednej panorámy. Parameter `dst` je ukazovateľ do CPU pamäte, kam má byť výsledok uložený. Táto funkcia je kompletne synchrónna a blokuje kým nie je skladanie a kopírovanie výsledku dokončené.

`get_output_image()`
: V prípade ak je potrebné získať výsledok asynchrónne, alebo v GPU pamäti, je možné použiť túto funkciu. Keďže môže vrátiť referenciu na výsledný obraz ešte pred dokončením skladania, je pred čítaním z tejto referencie nutné vykonať vlastnú synchronizáciu.

`get_input_stream()` a `get_input_stream()`
: Tieto funkcie slúžia na získanie gpu prúdov, ktoré možno využiť na vlastnú synchronizáciu v prípade využitia niektorých asynchrónnych funkcií poskytnutých týmto rozhraním.

5.2.5 Trieda `Cuda_stream`

Slúži ako RAII (Resource Acquisition Is Initialization)[2] obal pre ukazovateľ `CUstream_st*`, ktorý reprezentuje CUDA prúd. Zodpovedá

2. `https://en.cppreference.com/w/cpp/language/raii`

za jeho vytvorenie a korektné uvoľnenie. CUDA prúdy a ich využitie sú popísané v sekcii 5.7.

`get()`
: Vráti držaný ukazovateľ typu `CUstream_st*`.

`synchronize()`
: Synchronizuje s držaným prúdom. Synchronizácia je vykonaná volaním funkcie `cudaStreamSynchronize()`.

5.2.6 Trieda `Image_cuda`

Táto trieda reprezentuje obrazovú snímku uloženú v GPU pamäti. Obsahuje funkcie na získanie rôznych informácií o tomto snímku ako jeho rozmery, počet bytov potrebných na uloženie jedného riadku. Okrem toho je táto trieda zodpovedná za alokáciu a korektné uvoľnenie GPU pamäte využívanej na uloženie obrazových dát.

`get_width()` a `get_height()`
: Slúžia na získanie rozmerov snímku.

`get_bytes_per_px()`
: Vráti počet bytov potrebných na uloženie jedného obrazového bodu. V súčastnosti skladač pracuje len so snímkami uložených pomocou formátu obrazových bodov RGBA a preto táto funkcia vždy vráti veľkosť 4 byty.

`get_row_bytes()`
: Vráti počet bytov potrebných na uloženie jedného riadku obrazu. Výsledok je ekvivalentný ku $get\_width() \times get\_row\_bytes()$.

`get_pitch()`
: Vráti počet bytov medzi začiatkami po sebe následujúcich riadkov. Táto hodnota nemusí byť rovná `get_row_bytes()`, pretože riadky môžu z dôvodu pamäťového zarovnania medzi sebou obsahovať medzery.

5.3 Prehľad interných štruktúr a tried

Táto sekcia obsahuje krátky popis k štruktúram a triedam, ktoré sú v knižnici využívané len interne a nie sú súčasťou verejného rozhrania.

`Stitcher_impl`
: Táto trieda obsahuje reálnu implementáciu skladača. Rozhranie tejto triedy je identické s triedou `Stitcher`, ktorá obsahuje ukazovateľ na inštanciu tejto triedy a verejne sprístupňuje jej funkcie.

`Cam_stitch_context`
: Funkciou tejto triedy je uchovávať všetky potrebné informácie a dáta viazané ku jednej konkrétnej kamere. Obsahuje bloky pamäte na uloženie vstupného snímku a aj výsledok jej transformácie. Ďalej obsahuje parametre kamery a objektívu uložené v inštancii triedy `Cam_params`, parametre skladania a blendingu `Stitcher_properties` a CUDA prúd viazaný k danej kamere.

5.4 Inicializácia

Za účelom ušetrenia času pri skladaní je vhodné vykonať niektoré výpočty ešte pred zahájením skladania. V momente volania konštruktoru už sú k dispozícii informácie o všetkých kamerách a veľkosti výstupného obrazu. Vďaka tomuto je možné vykonať alokácie pamäte pre vstupné snímky, medzivýpočty a výsledná snímka dopredu a počas behu skladača už žiadne pamäťové alokácie neprebiehajú.

Pre každú kameru je takisto z poskytnutých informácií o rotácii a objektíve vypočítaná rotačná matica a ohraničujúca obdĺžniková oblasť, ktorú táto kamera pokrýva vo výstupnom snímku. Následne sa vyhľadajú prekryvy týchto oblastí medzi susednými kamerami.

5.4.1 Vyhľadávanie prekryvov

Keďže táto práca predpokladá kamerovú súpravu s rozložením horizontálneho prstenca, vyhľadávajú sa prekryvy kamier len v horizontálnom smere.

Pre každú kameru vypočítame tri uhly udávajúce horizontálny smer (anglicky sa takéto uhly označujú „yaw“) ľavého okraju, stredu a pravého okraju obrazu. Pre zjednodušenie výpočtu sú potom tieto uhly normalizované do intervalu $(-\pi, \pi]$. Za prekryv dvoch snímok berieme situáciu v ktorej ľavý okraj obrazu napravo sa nachádza medzi stredom a pravým okrajom obrazu naľavo.

Takto nájdené prekryvy sú potom uchované ako kolekcia štruktúr `Overlap`, ktoré obsahujú indexy ľavej a pravej kamery a horizontálne uhly začiatku a konca prekryvu.

5.5 Projekcia kamery

Transformácia vstupných snímok do rektilineárnej valcovej projekcie riadi funkcia `project_cam` triedy `Stitcher_impl`. Aby nebola transformácia počítaná zbytočne pre obrazové body ležiace mimo snímky zaznamenaného danou kamerou, sú najprv pomocou uhlov udávajúcich ľavý a pravý okraj zaznamenávaného obrazu vypočítané súradnice v cieľovej projekcii, ktoré ohraničujú oblasť do ktorej sa obraz kamery premietne. Hodnota obrazových bodov ležiacich v tejto oblasti je následne vypočítaná na grafickej karte pomocou funkcie `cuda_project_cam` implementovanej v súbore `project_cam.cu`.

5.5.1 Výpočet transformácie

V tejto podsekcii sú popísané detaily implementácie algoritmu popísaného v kapitole 4.4.

Na výpočet vektoru $\vec{v}$ musíme najprv určiť uhly φ a λ. Na ich výpočet je možné upraviť vzorce 2.4:

$$\begin{aligned} \varphi &= (\frac{h}{2} - y) \div \frac{h}{2} \times \frac{\pi}{2} \\ \lambda &= (x - \frac{w}{2}) \div \frac{w}{2} \times \pi \end{aligned} \tag{5.1}$$

Vektor $\vec{v}$ potom získame výpočtom súradníc bodu, ktorý leží na jednotkovej guli, ktorého „zemepisná“ šírka a dĺžka odpovedajú uhlom φ a λ.

$$\vec{v} = \begin{bmatrix} sin(\lambda) \times cos(\varphi) \\ sin(\varphi) \\ cos(\lambda) \times cos(\varphi) \end{bmatrix} \tag{5.2}$$

Za účelom zjednodušenia a urýchlenia následujúcich výpočtov je tento vektor následne transformovaný takou rotáciou, ktorá rotuje optickú

os kamery tak, aby ležala na osi z súradnicového systému. Táto transformácia je vykonaná rotačnou maticou, ktorú je možné pre každú kameru predpočítať pri inicializácii.

$$\vec{v\_rot} = \begin{bmatrix} cos(roll) & -sin(roll) & 0 \\ sin(roll) & cos(roll) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(pitch) & -sin(pitch) \\ 0 & sin(pitch) & cos(pitch) \end{bmatrix} \begin{bmatrix} cos(yaw) & 0 & sin(yaw) \\ 0 & 1 & 0 \\ -sin(yaw) & 0 & cos(yaw) \end{bmatrix} \vec{v} \quad (5.3)$$

Uhol θ medzi vektorom $\vec{v\_rot}$ a optickou osou kamery $\vec{u}$ je potom možné vypočítať pomocou známeho vzorca:

$$cos\theta = \frac{\vec{u} \cdot \vec{v\_rot}}{\|\vec{u}\| \|\vec{v\_rot}\|} \quad (5.4)$$

Keďže oba vektory sú jednotkové vektory (ich veľkosť je 1) a vektor $\vec{u}$ je navyše orientovaný v smere osi z, je tento vzorec možné zjednodušiť na $cos\theta = v\_rot_z$.

Po výpočte vzdialenosti r hľadaného bodu od stredu obrazu dosadením uhlu θ do vzorca 2.3 a korekcii skreslenia objektívu pomocou vzorca 4.2 je na určenie pozície potrebné vektor v_rot kolmo premietnuť na plochu kolmú od optickej osi kamery. Vďaka rotácii osi kamery na os z je možné tento priemet vykonať jednoducho vynulovaním zložky vektoru $\vec{v\_rot}$: $\vec{v\_proj} = \begin{bmatrix} v\_rot_x & v\_rot_y & 0 \end{bmatrix}'$.

Podľa postupu opísaného v kapitole 4.4 sú následne určené súradnice hľadaného bodu.

5.5.2 Vzorkovanie obrazových bodov vstupného snímku

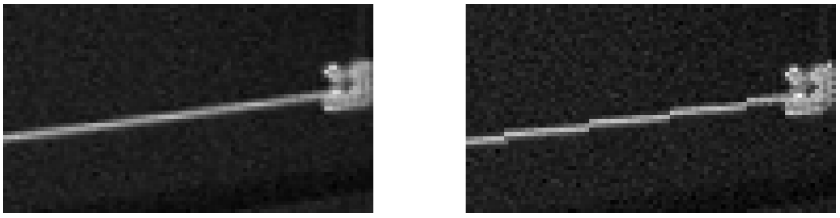
Súradnice obrazového bodu vypočítané v predchádzajúcej podsekcii sú vypočítané ako reálne čísla. Obrazové body sú však obvykle adresované diskrétne pomocou celých čísel. Jednoduchým riešením je súradnice zaokrúhliť na najbližšie celé číslo. Nevýhodou tohto riešenie je, že vo výslednom obraze potom môžu vznikať „zubaté“ hrany. Aby sa tomuto problému predišlo, je žiadúce využiť zložitejší spôsob vzorkovania, ktorý je schopný indexovať vzorky spojite pomocou interpolácie.

Interpolácia obrazových bodov je v 3D grafike často využívaná a preto sú grafické akcelerátory na túto úlohu dobre prispôsobené.

Na platforme CUDA je táto funkcionalita sprístupnená pomocou `cudaTextureObject_t` objektov[16].

Pre optimálne využitie tejto funkcionality sú snímky na grafickej karte uložené pomocou takzvaných CUDA polí (`cudaArray`) do pamäťovej oblasti prispôsobenej na skladovanie obrazových dát (Texture memory)[17]. K prístupom do tejto pamäte je využívaná špeciálna vyrovnávacia pamäť, ktorá je optimalizovaná pre dvojrozmernú lokalitu dát. Textúrové objekty obsahujú referenciu na toto pole, spolu s konfiguráciou, ktorá udáva spôsob ako sa majú tieto dáta interpretovať. Lineárnej interpolácie je možné dosiahnuť nastavením parametru `filterMode` na hodnotu `cudaFilterModeLinear`. Toto nastavenie zároveň vyžaduje aby parameter `readMode` bol nastavený na hodnotu `cudaReadModeNormalizedFloat`, čo spôsobí, že hodnoty obrazových bodov budú predávané ako reálne čísla v intervale $[0.0, 1.0]$.

Hodnoty obrazových bodov je potom možné získavať aj s použitím reálnych súradníc pomocou funkcie `tex2D<float4>()`. Na získanie 8-bitovej celočíselnej hodnoty jednotlivých zložiek je kvôli normalizácii nutné tieto zložky vynásobiť číslom 255. Dopad interpolácie pri vzorkovaní je možné vidieť na obrázku 5.1.



Obr. 5.1: Porovnanie vzorkovania pomocou interpolácie (na ľavo) a zaokrúhľovania (na pravo). Zväčšené pre zvýraznenie.

5.6 Kombinácia

Po transformácii snímok dochádza k ich kombinácii volaním funkcie `Stitcher::stitch()`. Táto kombinácia prebieha v dvoch krokoch: najprv sa do cieľového snímku skopírujú časti snímok ktoré nie sú súčasťou žiadneho prekrytia a následne sú vypočítané plynulé prechody v oblastiach prekrytia.

5.6.1 Kopírovanie neprekrývajúcich sa častí

Túto časť kombinácie vykonáva funkcia `copy_non_overlapping()`. Keďže sa predpokladá prstencové rozmiestnenie kamier, je možné usúdiť, že k prekryvom dochádza len v horizontálnom smere. Na určenie neprekrývajúcej sa oblasti preto stačia horizontálne uhly určujúce začiatok a koniec oblastí prekryvov, ktoré boli vypočítané počas inicializácie. V prípade ak sa obraz z nejakej strany neprerýva vôbec, je oblasť na kopírovanie určená pomocou horizontálneho zorného uhlu kamery a jej optickej osi.

Samotné kopírovanie je následne vykonávané pomocou funkcie `copy_image()`. Táto funkcia skrýva implementačné detaily uloženia snímku v pamäti. Vyjadrenie oblastí pomocou súradníc a rozmerov udávaných v obrazových bodoch je s uvážením použitého formátu obrazových bodov a pamäťovými medzerami medzi riadkami (pitch) prepočítané na pamäťové úseky. Tie sú následne kopírované pomocou funkcie `cudaMemcpy2D()`.

5.6.2 Výpočet plynulých prechodov

Na výpočet plynulých prechodov existuje viac algoritmov. V rámci tejto práce sú implementované algoritmy jednoduchého lineárneho blendingu a viacpásmového blendingu. Za účelom umožnenia užívateľom medzi týmito algoritmami prepínať a uľahčenie prípadného doplnenia iného algoritmu v budúcnosti, je na výpočet prekryvov použitá abstraktná trieda `Blender`, ktorá slúži ako rozhranie. Konkrétne algoritmy sú potom implementované ako podtriedy. Nasleduje stručný popis tohto rozhrania.

`Blender()`
: Konštruktor pomocou ktorého sú blenderu predané informácie o prekryvoch.

`blend_overlaps()`
: Slúži na výpočet prechodov. Parametrami tejto funkcie sú cieľová snímka a objekty typu `Cam_stitch_ctx` ktoré obsahujú premietnuté vstupné snímky.

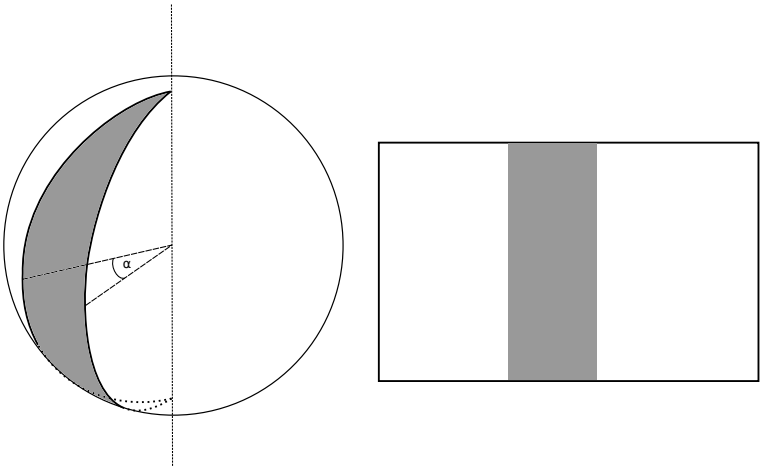
`submit_image()`
: Niektoré algoritmy (napr. viacpásmový blending) môžu vyko-

nať nejaké výpočty už v čase, keď majú ešte len jedna snímka z oblasti prekryvu. Táto funkcia slúži na signalizáciu, že už je možné naplánovať a spustiť tieto výpočty pre vstupná snímka s daným indexom.

5.6.3 Jednoduchý lineárny blending

Algoritmus jednoduchého lineárneho blendingu je implementovaný triedou `Feather_blender`. Výpočet prebieha na grafickej karte vo funkcii `kern_blit_overlap`.

Zvislá obdĺžniková oblasť prekryvu v cieľovej oblasti v ekvidistantnej valcovej projekcii odpovedá výseči povrchu gule. Šírka tejto oblasti vizuálne klesá smerom k pólom gule (znázornené na obrázku 5.2). Kvôli tomuto javu je na dosiahnutie vizuálne rovnomerne širokej prechodovej oblasti potrebné vykonávať korekciu jej šírky v závislosti na vertikálnom uhle. Zmena šírky odpovedá pomeru polomeru kruhovej výseče s daným vertikálnym uhlom k polomeru výseče s nulovým vertikálnym uhlom. Upravenú šírku je teda možné vypočítať ako $width\_corr = width \times \frac{1}{cos(\varphi)}$. Keďže maximálna šírka prechodu je daná šírkou oblasti prekryvu, je na ďalšie výpočty použité maximum z týchto hodnôt.



Obr. 5.2: výseč na povrchu gule odpovedajúca vertikálnej obdĺžnikovej oblasti v ekvidistantnej valcovej projekcii

V prípadoch keď je šírka prechodovej oblasti menšia ako šírka oblasti prekryvu môže byť žiadúce mať možnosť prechodovú oblasť jemne posúvať. Maximálna miera posunu stredu prechodu závisí od šírky prekryvu a šírky prechodu. Jeho horizontálna súradnica musí spadať do intervalu $[\frac{seam\_width}{2}, overlap\_width - \frac{seam\_width}{2}]$. Vďaka tomuto obmedzeniu sa dá posun použiť súčasne s korekciou šírky opísanou v predchádzajúcom odstavci - s rastúcou šírkou prechodu je jeho stred plynule posúvaný do stredu prechodovej oblasti.

Pomocou vypočítaného stredu a šírky prechodu je potom možné pre horizontálnu súradnicu x vypočítať váhu udávajúcu podiel ľavého a pravého snímku:

$$\begin{aligned} seam\_start &= seam\_center - \frac{seam\_width}{2} \\ weight &= max(0, min(1, \frac{x - seam\_start}{seam\_width})) \end{aligned} \tag{5.5}$$

Pomocou tejto váhy je potom možné vypočítať hodnotu výsledného obrazového bodu násobením po zložkách $result = left \times weight + right \times (1 - weight)$.

5.6.4 Viacpásmový blending

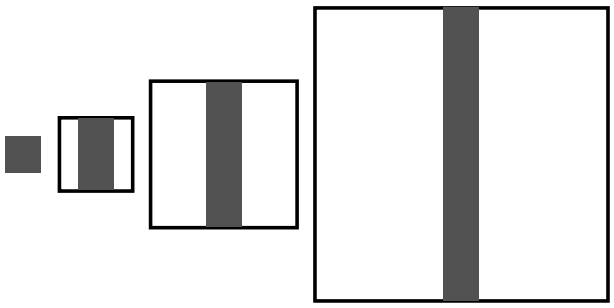
Viacpásmový blending je implementovaný triedou `Multiband_blender`. Pri konštrukcii inštancie tejto triedy je pre každý prekryv vytvorený objekt `Overlap_pyramid`, ktorý obsahuje laplacove pyramídy pre ľavý a pravý obraz prekryvu. Okrem toho obsahuje trieda ešte pomocnú pyramídu s rozmerom najväčšieho prekryvu, ktorá sa používa na medzivýpočty.

Volaním funkcie `submit_image()` je možné pre vstupnú snímku vypočítať obsah jednotlivých poschodí pyramíd pre jeho oblasti prekryvu. Pri výpočte výsledných plynulých prechodov volaním funkcie `blend_overlaps()` sú tieto pyramídy využité na výpočet jednotlivých poschodí pomocnej pyramídy. Každé poschodie tejto pyramídy je tvorené lineárnym prechodom medzi odpovedajúcimi poschodiami pyramíd ľavého a pravého obrazu. Tieto prechody sú počítané funkciou `cuda_feather_simple()`. Jedná sa o jednoduchšiu verziu algoritmu

popísaného v predchádzajúcej sekcii, ktorá nevykonáva korekciu šírky prechodu.

Šírka tohto prechodu je namiesto toho konštantná a nezávisí od vertikálneho uhlu a ani od počítaného poschodia pyramídy. Táto šírka je volená tak, aby na poschodí s najmenším rozmerom pokrýval prechod celý obraz. Keďže každé následujúce poschodie má dvojnásobný rozmer, je držaním konštantnej šírky prechodu zaručené, že pre poschodia obsahujúce vyššie frekvencie je prechod ostrejší (viz obr. 5.3). Aby boli stredy prechodov na jednotlivých poschodiach medzi sebou zarovnané je prechod umiestnený vždy do stredu. Pri využití tohto algoritmu preto nie je možné prechod posúvať.

Výsledný prechod je potom získaný rekonštrukciou takto vypočítanej pyramídy.



Obr. 5.3: Znázornenie konštantnej šírky prechodu. Pri poschodiach s väčším rozmerom zaberá prechod menšiu časť obrazu.

5.6.5 Implementácia Laplacovej pyramídy

Laplacova pyramída používaná pri viacpásmovom blendingu je implementovaná pomocou triedy `Pyramid`. Táto trieda obsahuje pole obrazov typu `Image_cuda`, ktorých počet odpovedá zvolenému počtu poschodí. Prvý z týchto obrazov má plný rozmer, každý následujúci má oproti predchádzajúcemu rozmer polovičný.

Pri konštrukcii jednotlivých poschodí a pri rekonštrukcii pôvodného obrazu dochádza k operáciam zmenšovania a zväčšovania obrazu. Zmenšenie obrazu na polovičnú veľkosť je vykonávané pomocou funkcie `cuda_downsample()`, ktorá do cieľového obrazu jednoducho

zapíše každý druhý obrazový bod. Naopak na zväčšenie slúži funkcia `cuda_upsample()`, ktorá do cieľového obrazu zapíše každý obrazový bod dva krát.

Takéto zmenšovanie obrazu zníži jeho vzorkovaciu frekvenciu na polovicu. Podľa Nyquist-Shannonovho teorému môže obraz obsahovať len frekvencie rovné alebo menšie ako polovica vzorkovacej frekvencie[18]. V prípade ak vstupný obraz obsahoval frekvencie vyššie, mohlo by týmto spôsobom podvzorkovania dôjsť k Moirého efektu. Z tohto dôvodu je preto nutné pred zmenšením z pôvodného obrazu tieto frekvenčné zložky odstrániť. Zväčšenie obrazu zdvojením obrazových bodov môže do obrazu tiež zaviesť vysokofrekvenčné zložky, čo je ľahko pozorovateľné pri opakovanom zväčšovaní touto metódou - v obraze vznikajú „štvorčeky“ s ostrou hranou. Na elimináciu tohto efektu je preto potrebné vysokofrekvenčné zložky odstrániť aj po každom zväčšení.

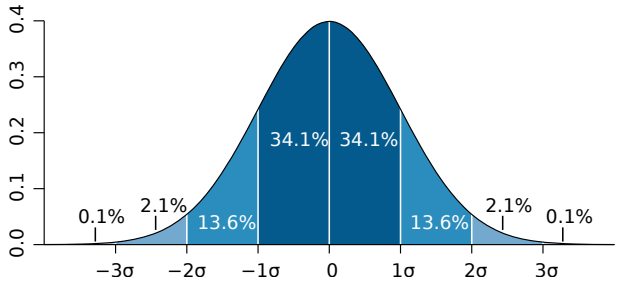
Na odstránenie vyšších frekvencií z obrazu je možné využiť Gaussovo rozmazanie[19]. Toto rozmazanie je možné implementovať ako konvolúciu obrazu s konvolučným jadrom, ktoré je tvorené Gaussovou funkciou. Toto jadro je separabilné, čo znamená že dvojrozmernú konvolúciu je možné vykonať pomocou dvoch menších jednorozmerných jadier[20]. Na výpočet konvolučného jadra slúži trieda `Gaussian_kernel`, ktorá počíta hodnoty jednotlivých bodov pomocou vzorca 5.6.

$$g(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{5.6}$$

Táto funkcia má maximum v bode 0 a s rastúcou vzdialenosťou od tohto bodu jej hodnota klesá. Rozmer konvolučného jadra je volený tak, aby hodnoty ktoré spadajú mimo jeho rozsah boli už zanedbateľne malé. Po vypočítaní všetkých bodov jadra sú tieto body normalizované tak, aby sa ich súčet bol rovný 1. Cieľom filtrovania obrazu je odstrániť frekvenčné zložky ktoré nie sú reprezentovateľné v obraze polovičnej veľkosti. Reprezentovateľný rozsah frekvencií je v dôsledku Nyquist-Shannonovho teorému pri takomto zmenšení redukovaný na polovicu. Regulovanie frekvenčného rozsahu v ktorom dochádza k útlmu je možné nastavením štandardnej odchýlky σ. Pri hľadaní správnej hodnoty je možné použiť vzorec 5.7, ktorý udáva frekvenčnú odozvu tohto filtra.

$$\widehat{g}(f) = e^{-\frac{f^2}{2\sigma_f^2}}$$
$$\sigma_f = \frac{1}{2\pi\sigma} \tag{5.7}$$

Keďže nosič (časť definičného oboru, kde funkcia nadobúda nenulovú hodnotu) Gaussovej funkcie je nekonečne veľký, nie je možné nežiadúce frekvencie z obrazu odstrániť úplne. Pri odchýlkach väčších ako 2σ už ale dochádza k dostatočnému útlmu (ukážka na obrázku 5.4).



Obr. 5.4: Dopad odchýlky na Gaussovu funkciu[3]

Rozmazanie je implementované funkciou `cuda_gaussian_blur()`. Táto funkcia vykoná konvolúcie v horizontálnom a vertikálnom smere pomocou funkcií `kern_gauss_blur_row()` a `kern_gauss_blur_col()`. Použité je jadro konvolúcie s veľkosťou 7 bodov a parametrom $\sigma = 1.4$.

Obsah jednotlivých poschodí pyramídy je možné vypočítať zo vstupných obrazových dát pomocou funkcie `construct_from()`. Pri výpočte poschodia je najskôr vytvorený obraz polovičnej veľkosti. Tento polovičný obraz je potom naspäť zväčšený. Obsah poschodia je potom vypočítaný ako rozdiel tohto obrazu oproti obrazu pôvodnému. Na výpočet tohto rozdielu slúži funkcia `cuda_subtract_images()`. Keďže pracujeme s obrazmi ktorých body sú reprezentované pomocou 8 bitových hodnôt bez znamienka (rozsah $[0, 255]$) je žiadúce sa vyhnúť záporným číslam vo výsledku. Tohto je možné docieliť

3. `https://commons.wikimedia.org/wiki/File:Standard_deviation_diagram.svg`

pripočítaním čísla 128 k rozdielu. Tento posun nám umožní reprezentovať výsledky v intervale $[-128, 127]$. Tento interval síce nestačí na uchovanie rozdielu ľubovoľných obrazov, ale na použitie v Laplacovej pyramíde postačuje, pretože rozmazanie, okrem extrémnych prípadov ako jeden obrazový bod s maximálnou intenzitou obklopený bodmi s minimálnou intenzitou, nevytvorí príliš veľké rozdiely. Pre posledný (najmenší) level sa rozdiel už nepočíta, namiesto toho sa zmenšený obraz len uloží.

Pôvodný obraz je možné z pyramídy rekonštruovať pomocou funkcie `reconstruct_to()`. Pri rekonštrukcii dochádza, začínajúc posledným najmenším poschodím, k postupnému zväčšovaniu obrazu. Po každom zväčšení sa pripočíta rozdiel z vyššieho poschodia, ktorý obsahuje vysokofrekvenčné zložky. Na sčítanie obrazov slúži funkcia `cuda_add_images()`. Po pripočítaní rozdielu z každého poschodia je výsledkom bezstrátová rekonštrukcia pôvodného obrazu.

Každé poschodie pyramídy má oproti predchádzajúcemu polovičný rozmer, čo znamená, že rozmer najväčšieho z nich by mal byť deliteľný 2^n, kde n je počet poschodí. Rozmer prechodových oblastí môže ale nadobúdať ľubovoľné hodnoty a preto je potrebné vyriešiť aj prípady neideálnych rozmerov. Na vyriešenie tohto problému je najskôr rozmer zaokrúhlený na najbližší menší alebo rovný násobok 2^n. Výpočet pyramídy potom prebieha so vstupom orezaným na tento rozmer s výnimkou najväčšieho poschodia, ktoré obsahuje zvyšnú časť pôvodného obrazu.

5.7 Súbežnosť pamäťových prenosov a výpočtov

Aby bolo možné vykonávať výpočty nad vstupnými obrazovými dátami pomocou grafickej karty, je najprv nutné tieto dáta skopírovať do GPU pamäte. Toto kopírovanie vzhľadom k obmedzenej šírke pásma zbernice PCIe predstavuje nezanedbateľnú réžiu - pri bežnom použití sa pohybuje okolo 1.56 ms až 3 ms pre každú vstupnú snímku (viz kapitola 8). Grafické karty firmy Nvidia umožňujú simultánne vykonávať výpočty a pamäťové prenosy. Prenášaním dát pre následujúci výpočet, kým prebieha predchádzajúci umožňuje „skrývať“ túto réžiu pamäťových prenosov.

Na umožnenie súbežnosti sú na platforme CUDA operácie radené do frônt. Tieto fronty sú označované ako prúdy (stream)[21]. Operácie v rámci jedného prúdu sú vykonávané sekvenčne po sebe. Ak sú operácie v dvoch rôznych prúdoch, ich poradie vykonania nie je deterministické a môžu byť vykonávané aj simultánne. Aby bolo možné v prúde využiť výsledky vypočítané iným prúdom, je potrebné sa najprv uistiť, že prúd, ktorý tieto dáta vyprodukoval, už dobehol. Za týmto účelom je poskytnutá funkcia `cudaStreamSynchronize()`, ktorá blokuje výpočet, kým v danom prúde nie sú dokončené všetky operácie.

Pri procese skladania panorámy sú vstupné snímky pred samotnou kombináciou spracovávané nezávisle od seba. Preto má zmysel pre každú kameru vytvoriť samostatný prúd. Tento prúd je uložený v objekte `Cam_stitch_context` a je používaný pre pamäťový prenos vstupného snímku, transformáciu do výstupnej projekcie a v prípade použitia viacpásmového blendingu aj na výpočet Laplacových pyramíd. Tieto operácie sú do fronty zaradené už po volaní funkcie `submit_input_image()` vďaka čomu sú vykonávané hneď ako je vstupná snímka pre danú kameru k dispozícii.

Ku kombinácii dochádza volaním funkcie `stitch()`. Kombinácia snímok prebieha v samostatnom prúde nazvanom `out_stream`. Na zaistenie korektnosti prístupu do pamäte je pred čítaním každej snímky výpočet synchronizovaný s prúdom príslušnej kamery. Tento výstupný prúd je použitý aj na stiahnutie výsledku z gpu pamäte, čím sa zaistí, aby stiahnutie prebehlo až po dokončení výpočtu.

5.8 Integrácia do programu UltraGrid

Ultragrid[4] je softvér vyvíjaný pod slobodnou licenciou BSD, ktorý sa zameriava na živé prenosy videa a zvuku pomocou bežne dostupného spotrebiteľského hardvéru. Medzi výhody tohto softvéru patrí podpora vysokého rozlíšenia a nízkeho oneskorenia prenosu[22].

Tento program podporuje množstvo vstupných a výstupných zariadení a algoritmov kompresie. Táto podpora často vyžaduje závislosti na knižniciach tretích strán, ktoré však často nie sú podporované na všetkých platformách pre ktoré je UltraGrid vyvíjaný. Z tohto

4. `http://www.ultragrid.cz/`

dôvodu je štruktúra tohto programu modulárna, čo umožňuje pri preklade zapínať a vypínať podporu určitých funkcií. Skladač video panorámy je preto do tohto programu integrovaný ako modul zaznamenávania videa s názvom `gpustitch`, implementovaný v súbore `src/video_capture/gpustitch.cpp`.

5.8.1 Rozhranie modulu

Komunikácia s modulmi na záznam videa prebieha v programe Ultragrid pomocou rozhrania tvoreného následujúcimi funkciami:

`probe()`
: Slúži na predanie informácií o module programu UltraGrid. V prípade modulu `gpustitch` sa jedná o jeho názov a krátky popis zobrazovaný v nápovede.

`init()`
: Pomocou tejto funkcie je vykonaná inicializácia modulu. Modulu sú predané parametre z príkazového riadku. Pre uchovanie dát potrebných na beh môže modul nastaviť ukazovateľ `state`. Tento ukazovateľ je potom predaný ako parameter pri volaní následujúcich funkcií.

  Modul `gpustitch` pri inicializácii spracuje parametre z príkazového riadku, prečíta parametre kamerovej súpravy zo súboru a pomocou týchto hodnôt inicializuje inštanciu skladača. Na uchovanie tohto stavu je alokovaná štruktúra `vidcap_gpustich_state` a ukazovateľ na ňu je zapísaný do výstupného parametra `state`.

`done()`
: Po ukončení práce s modulom je volaná táto funkcia. Slúži na uvoľnenie zdrojov.

`grab()`
: Táto funkcia slúži na prečítanie následujúceho snímku. V prípade modulu `gpustitch` táto funkcia zodpovedá za zavolanie funkcie `stitch()` a stiahnutie výsledného snímku.

5.8.2 Získavanie vstupných snímok

Aj keď je skladač integrovaný do programu UltraGrid ako modul na zaznamenávanie videa, slúži len na skladanie už zaznamenaného videa. Samotné skladanie je vykonávané pomocou jednej alebo viacerých inštancií ostatných zaznamenávacích modulov, ktoré sú modulu `gpustitch` predané pomocou parametrov zadaných na príkazovom riadku.

Zaznamenávanie môže prebiehať v dvoch módoch. V prvom móde je obraz pre každú kameru zaznamenávaný samostatnou inštanciou zaznamenávacieho modulu. Druhý mód slúži na zaznamenávanie štyroch kamier súčasne jednou inštanciou, tak ako bolo opísané v kapitole 3.6. V oboch módoch prebieha záznam z každej inštancie v samostatnom vlákne.

Pre každé vlákno je vytvorená samostatná inštancia štruktúry `grab_worker_state`, ktorá slúži na uchovanie inštancie zaznamenávacieho modulu, vyrovnávacej pamäte pre medzivýpočty a ďalšieho stavu. Tieto vlákna vykonávajú funkciu `grab_worker()`. V tejto funkcii prebieha čítanie snímok zo zaznamenávacieho modulu, následná kontrola ich formátu a ich predanie skladaču. U snímok sa kontroluje ich rozmer, ktorý musí odpovedať rozmeru uvedenému v parametroch kamery. V prípade ak je formát obrazových bodov odlišný od formátu RGBA je počas kroku kontroly zvolená vhodná konverzná funkcia a ukazovateľ na ňu je uložený do položky `conv_func` v štruktúre `grab_worker_state`.

5.8.3 Konverzia formátu obrazových bodov

Zatiaľ čo knižnica na skladanie panorámy pracuje s obrazmi, ktorých obrazové body sú reprezentované ako 8-bitové RGBA hodnoty, v programe UltraGrid sú často používané rôzne iné formáty. Zaznamenané snímky môže byť preto nutné pred predaním do skladača konvertovať do požadovaného formátu.

Za účelom minimalizácie výkonnostného dopadu konverzie, je konverzia vykonávaná na grafickej karte. Aktuálne je podporovaná konverzia pre formáty RGB a UYVY pomocou funkcií implementovaných v súbore `src/utils/cuda_pix_conv.cu`.

Predávanie vstupných snímok skladaču je riadené pomocou funkcie `upload_to_cuda_buf()`. Táto funkcia v prípade nutnosti konverzie snímku najprv skopíruje do gpu pamäte `tmp_in_frame` uloženej v štruktúre `grab_worker_state` a následne volá konverznú funkciu. Celý tento proces prebieha s použitím vstupného prúdu pre danú kameru, získaného pomocou funkcie `get_input_stream()`.

Podobne je možné konvertovať aj výstupné snímky z formátu RGBA do formátov RGB a UYVY. Užívateľ môže požadovaný výstupný formát zadať ako parameter na príkazovom riadku, podľa ktorého sa pri inicializácii vyberie vhodná konverzná funkcia. Táto konverzia je iniciovaná vo funkcii `grab()` a prebieha vo výstupnom prúde.

5.8.4 Synchronizácia zaznamenávajúcich vlákien

Prístupy do pamäte a synchronizácia výpočtov je v skladači interne vyriešená pomocou CUDA prúdov a preto je možné volať funkcie na predanie vstupných snímok a funkciu `stitch()` z rôznych vlákien. Pre korektný beh je potrebné zaručiť, aby následujúce snímky boli skladaču predané až potom, ako bola zavolaná funckia `stitch()` pre predchádzajúce snímky. Taktiež je potrebné zaistiť, aby funckia `stitch()` bola volaná až potom, ako bolo iniciované predanie snímky všetkými zaznamenávajúcimi vláknami.

Synchronizácie je možné docieliť udržiavaním počtu poskladaných snímok a počtov predaných snímok pre každé vlákno. Každé zaznamenávacie vlákno potom pred predaním snímky skladaču čaká, kým sa počet poskladaných snímok nerovná počtu predošle predaných snímok. Naopak funkcia `grab()` zas pred volaním iniciáciou skladania čaká, kým počet predaných snímok pre každé vlákno nie je väčší ako počet už poskladaných snímiek. Prístup k týmto počtom je chránený s využitím zámkov `std::mutex` a čakanie je implementované pomocou `std::condition_variable`.

5.9 Zhrnutie

V tejto kapitole bola opísaná implementácia knižnice na skladanie panoramatického videa. Algoritmy popísané v kapitole 4 boli prebraté

detailnejšie, vrátane technických detailov ich implementácie. Súčasťou kapitoly je aj popis verejného rozhrania tejto knižnice. Na záver bola opísaná integrácia tejto knižnice do programu UltraGrid.

6 Implementácia zobrazovania

Táto kapitola sa zaoberá návrhom a implementáciou vykresľovania 360° videa. Pre dosiahnutie prirodzeného vzhľadu je očakávané, že vykreslený obraz bude v rektilineárnej projekcii. Implementácia bude umožňovať užívateľovi meniť smer pohľadu.

Pre podporu klasických spôsobov vykresľovania videa na monitor ako aj vykresľovanie do 3D headsetu bude zobrazovanie v programe UltraGrid implementované ako dvojica zobrazovacích modulov *pano_gl* a *openxr_gl*.

Táto kapitola sa zaoberá implementáciou modulu *pano_gl*, ktorý slúži na zobrazovanie v klasickom okne na monitore a na ovládanie využíva počítačovú myš. Keďže spacovanie obrazu je u oboch modulov prevažne rovnaké, abstrakcie ktoré sú popísané v tejto kapitole sú využité aj v module *openxr_gl*. Pri implementácii je využívané rozhranie OpenGL, ktoré je v programe UltraGrid už využívané pre zobrazovanie klasického videa.

6.1 Princíp vykresľovania

Transformáciu do rektilineárnej projekcie by bolo možné implementovať podobným spôsobom ako transformácie pri skladaní obrazu popísané v kapitole 5.5. S použitím rozhrania OpenGL je však jednoduchšie využiť trojrozmerné perspektívne vykresľovanie.

V sekcii 2.4 bolo spomenuté, že ekvidistantná valcová projekcia je často využívaná na zobrazovanie svetovej mapy. Podobne, ako je možné vykresľovať glóbus použitím trojrozmerného modelu gule, na ktorý je aplikovaná textúra, môžeme na guľu aplikovať vstupnú snímku ako textúru takým spôsobom, aby každému vrcholu ktorý túto guľu tvorí prislúchali textúrové súradnice odpovedajúce „zemepisnej šírke a dĺžke“ tohto vrcholu.

Po konštrukcii takéhoto modelu stačí už len umiestniť kameru do stredu takejto gule a scénu vykresliť štandardnými metódami, ktoré sa v počítačovej grafike bežne využívajú na vykresľovanie trojrozmerných scén.

6.2 Trieda `GlProgram`

V modernom OpenGL je vykresľovanie riadené programovateľnou pipeline. Jednotlivé časti tejto pipeline sú programované v jazyku GLSL (OpenGL Shading Language) a nazývajú sa shader programy. Prekladom týchto shader programov vznikajú shader objekty, ktoré sú potom skombinované do jedného program objektu[23]. Po aktivácii tohto objektu je vykresľovanie vykonávané s využitím shader programov z ktorých bol program objekt zostavený.

Trieda `GlProgram` je zodpovedná za zostavenie programového objektu a korektné uvoľnenie zdrojov po dokončení práce s daným programom.

6.2.1 Vertex a fragment shader

Na vykresľovanie 360° videa je použitý program obsahujúci vertex a fragment shader. Vertex shader slúži na spracovanie jednotlivých vrcholov. Použitý vertex shader je pomerne jednoduchý - jedinou operáciou, ktorú vykonáva, je vynásobenie vektoru obsahujúceho pozíciu vrcholu maticou, ktorá vykoná perspektívnu projekciu a rotáciu. Konštrukciou tejto matice sa bude zaoberať podsekcia 6.6.1. Textúrové súradnice sú bez zmeny predané ďalším krokom pipeline.

Fragment shader sa používa na výpočet konkrétnych hodnôt obrazových bodov. Shader použitý pri vykresľovaní gule je tiež veľmi jednoduchý - jedinou operáciou je prečítanie vzorky textúry na textúrových súradniciach získaných ako vstupný parameter z predchádzajúceho kroku pipeline.

6.3 Trieda `Model`

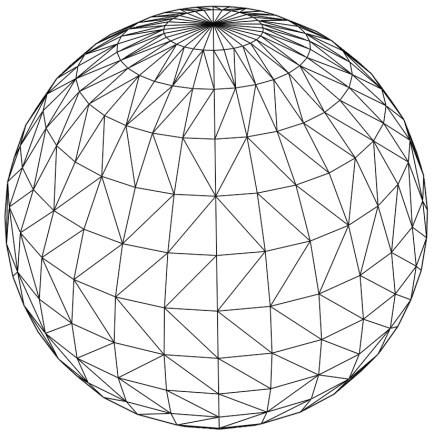
Na uľahčenie práce s rozhraním OpenGL som implementoval niekoľko tried, ktoré slúžia ako RAII obaly pre OpenGL prostriedky a zároveň obsahujú pár pomocných funkcií, ktoré uľahčia prácu s nimi.

Jednou z týchto tried je trieda `Model`, ktorá reprezentuje 3d model. Obsahuje OpenGL prostriedky typu vertex buffer objekt, ktorý obsahuje súradnice jednotlivých vrcholov modelu spolu s ich textúrovými súradnicami, vertex array object, ktorý obsahuje informáciu ako majú byť tieto dáta interpretované a element array buffer objekt, ktorý

obsahuje poradie vrcholov použité pri vytváraní trojuholníkov. Táto trieda ďalej obsahuje funkciu `render()`, ktorá tieto objekty aktivuje (bind) a vykreslí. Inštancie tejto triedy je možné vytvoriť pomocou funkcie `get_quad()`, ktorá vráti model obsahujúci obdĺžnik tvorený dvomi trojuholníkmi, alebo funkciou `get_spehere()`, ktorá generuje trojrozmerný model gule.

6.3.1 Generovanie modelu gule

Trojrozmerný model gule je možné generovať vytvorením vrcholov na priesečníkoch „poludníkov“ a „rovnobežiek“ a následným vytvorením trojuholníkov medzi vrcholmi ležiacimi na na susedných „rovnobežkách“. Pre lepšiu predstavu je drátový model takto vytvorenej gule znázornený na obr 6.1.



Obr. 6.1: Drôtený model gule

Spočiatku sa môže zdať, že trojuholníky pri „póloch“ gule vyžadujú špeciálny prístup pretože všetky zdieľajú jeden bod. V praxi je ale jednoduchšie tento prípad vyriešiť rovnakým spôsobom ako zvyšok gule - pól je tvorený množinou bodov, ktoré ležia na kruhovom výseku s nulovým polomerom. Súradnice vrcholov je možné generovať využitím dvoch vnorených cyklov, ktoré postupne iterujú celým rozsahom vertikálneho a horizontálneho uhlu v rovnomerne veľkých

krokoch. Pozíciu bodu je potom možné z týchto uhlov vypočítať pomocou trigonometrických funkcií rovnakým spôsobom ako prebiehal výpočet vektoru v kapitole 5.5.1. Tieto uhly je zároveň možné využiť aj na výpočet textúrových súradníc pomocou vzorca ekvidistantnej valcovej projekcie 2.4.

6.4 Trieda `Texture`

Na uľahčenie práce s OpenGL textúrami bola implementovaná trieda `Texture`, ktorá sa stará o korektné vytvorenie a uvoľnenie OpenGL textúrových objektov. Okrem toho implementuje funkcie na načítanie obrazových dát textúry z štruktúry `video_frame`, ktorá v programe UltraGrid reprezentuje jednu obrazovú snímku videa.

6.5 Konverzia formátu obrazových bodov

Obrazové snímky videa sú v programe UltraGrid často reprezentované pomocou schémy reprezentácie farieb YUV. Rozhranie OpenGL však túto schému nepodporuje priamo a predpokladá využitie schémy RGB. Pred použitím vstupného snímku môže byť preto nutné vykonať konverziu formátu obrazových bodov snímku.

V programe UltraGrid už existuje v zobrazovacom module `gl` fragment shader, ktorý slúži na správne vykresľovanie YUV textúr. Keďže OpenGL ponúka možnosť vykresľovať do textúr, je možné tento shader použiť na konverziu YUV textúry do novej RGB textúry.

Konverzia je implementovaná triedou `Yuv_convertor`. Inštancia tejto triedy zostaví `GlProgram` obsahujúci fragment shader vykonávajúci konverziu a jednoduchý vertex shader, ktorý súradnice vrcholov vôbec nemení. Cieľ vykresľovania je v OpenGL volený pomocou frame buffer objektov. Predvolený frame buffer sa štandardne zobrazuje na obrazovku, ale OpenGL ponúka mechanizmus, pomocou ktorého je možné vytvoriť nový frame buffer objekt, po aktivácii ktorého je vykresľovanie vykonávané do viazanej textúry. Trieda `Yuv_convertor` sa stará o korektné vytvorenie takéhoto frame buffer objektu a o prekreslenie vstupnej YUV textúry pomocou zostaveného programu do viazanej RGB textúry.

6.6 Trieda `Scene`

Trieda `Scene` kombinuje triedy opísané v predchádzajúcich sekciách a poskytuje jednoduché rozhranie na vykreslenie vstupného video snímku s požadovanou rotáciou. Medzi zodpovednosti tejto triedy patrí konštrukcia a inicializácia používaných OpenGL objektov, výpočet perspektívnej a rotačnej matice a v prípade potreby aj konverzia formátu obrazových bodov vstupného snímku.

6.6.1 Perspektívna a rotačná matica

Pri vykresľovaní trojrozmerných modelov je potrebné vypočítať, na aké súradnice obrazovky sa premietne každý vrchol modelu. Jedná sa teda o transformáciu z trojrozmerného súradnicového systému modelu do dvojrozmerného súradnicového systému obrazovky. Táto transformácia je vykonávaná pomocou matice, ktorou sa vo vertex shader programe vynásobí vektor udávajúci pozíciu každého vrcholu. Vhodne skonštruovanou maticou je možné vykonávať rotáciu vykresľovaných objektov a ich perspektívnu projekciu na obrazovku.

Matice je možné kombinovať ich vzájomným násobením a preto môžeme výslednú maticu skonštruovať ako kombináciu projekčnej matice a matice pohľadu. Na uľahčenie výpočtu je použitá knižnica glm[1]. Projekčnú maticu získame pomocou funkcie `glm::perspective`, ktorej argumentmi sú zorný uhol a pomer veľkostí strán výstupného obrazu. Maticu pohľadu, ktorá je zodpovedná za rotáciu vykresľovaného modelu podľa smeru pohľadu získame modifikáciou jednotkovej matice funckiou `glm::rotate`. Výslednú maticu získame vynásobením týchto dvoch matíc.

6.7 Vytvorenie okna a inicializácia OpenGL kontextu

Na vytvorenie okna som sa rozhodol použiť knižnicu SDL[2]. Táto knižnica je v programe UltraGrid už využívaná zobrazovacím modulom `sdl` a umožňuje vytváranie okien s OpenGL kontextom a spracovávanie vstupu z klávesnice a myši na širokej škále rôznych operačných

1. `https://glm.g-truc.net/`
2. `https://www.libsdl.org/`

systémoch. Táto knižnica disponuje rozhraním v jazyku C, a preto som implementoval triedu `Sdl_window`, ktorá slúži ako RAII obal pre ukazovateľe reprezentujúce sdl okno a OpenGL kontexty s ním asociované.

6.7.1 Inicializácia okna

Pre začiatkom práce s knižnicou sdl je potrebné vykonať inicializáciu video subsystému. Toto je vykonané volaním `SDL_InitSubSystem()`. Aby bolo možné využívať moderné OpenGL je potrebné nastaviť požadovaný profil a verziu OpenGL kontextu ešte pred jeho vytvorením pomocou funkcie `SDL_GL_SetAttribute()`. Zvolil som core profil verzie 3.3, ktorý ponúka všetku potrebnú funkcionalitu. Samotné okno je vytvorené funkciou `SDL_CreateWindow()`, ktorá umožňuje nastaviť názov okna, jeho pozíciu a príznaky. Vo svojej implementácii používam príznak `SDL_WINDOW_OPENGL`, ktorý určuje že sa bude používať rozhranie OpenGL a príznak `SDL_WINDOW_RESIZABLE`, ktorý umožňuje užívateľovi meniť veľkosť vytvoreného okna. Pred využitím OpenGL je potrebné ešte vytvoriť OpenGL kontext zviazaný s oknom. Toto je vykonané pomocou funkcie `SDL_GL_CreateContext`. Na využitie OpenGL funkcií zavedených v OpenGL verziách starších ako 1.1 je potrebné získať funkčné ukazovatele na tieto funkcie za behu programu[24].

6.7.2 Inicializácia OpenGL funkčných ukazovateľou

Pred použitím funkcií moderného OpenGL je potrebné na tieto funckie získať ukazovatele. Tento proces je platformovo závislý a je vykonávaný pomocou `GetProcAddress()` funkcie danej platformy[25]. Manuálne nahrávanie veľkého množstva funkcií by bolo neprehľadné a preto som použil knižnicu GLEW[3], ktorá tento proces automatizuje a zároveň skrýva rozdiely medzi platformami. Pomocou tejto knižnice je možné nahrať všetky funkcie jednoducho volaním `glewInit()`.

3. `http://glew.sourceforge.net/`

6.8 Implementácia zobrazovanieho modulu `pano_gl`

Zobrazovacie moduly sú v programe UltraGrid implementované v priečinku `src/video_display/`. V tejto sekcii bude opísaná implementácia modulu `pano_gl`, ktorý integruje skôr opísané triedy a umožňuje zobrazovanie 360° videa do okna na monitore.

6.8.1 Rozhranie zobrazovacích modulov

Zobrazovacie moduly, podobne ako moduly na získavanie videa, komunikujú so zvyškom programu UltraGrid pomocou rozhrania v jazyku C. Nasleduje krátky popis tohto rozhrania.

- `init()`
  Táto funkcia slúži na inicializáciu modulu. Ako parameter tejto funkcie sú predané argumenty z príkazového riadku. Moduly môžu v tejto funkcii získať rôzne zdroje a na uloženie svojho stavu môžu z tejto funkcie vrátiť jeden ukazovateľ, ktorý im potom bude predaný ako argument v ostatných funkciách rozhrania.

- `run()`
  Po inicializácii je displej „spustený“ volaním tejto funkcie. Táto funkcia blokuje po celú dobu behu modulu a väčšinou obsahuje nekonečný cyklus v ktorom prebieha vykresľovanie a spracovanie vstupu z klávesnice.

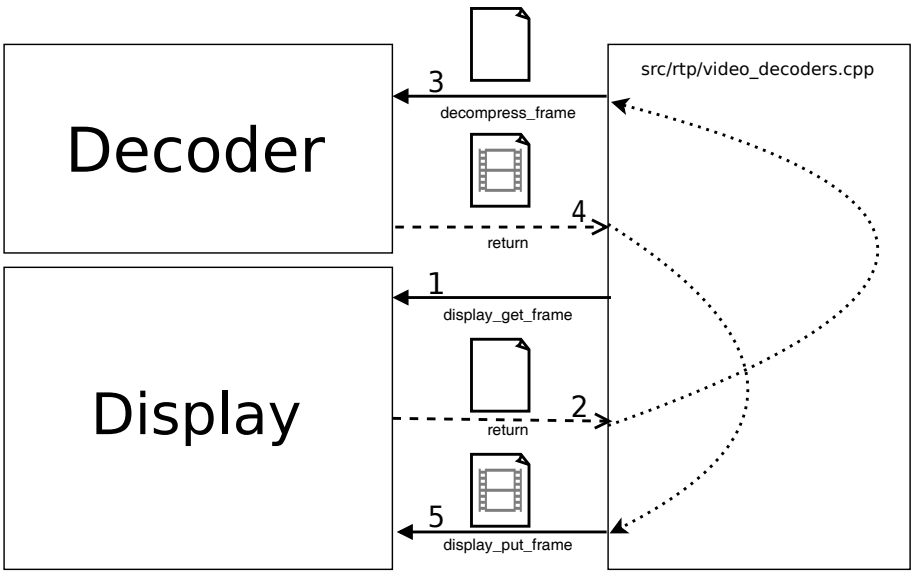
- `done()`
  Slúži na korektné uvoľnenie zdrojov po dokončení práce so zobrazovacím modulom.

- `getf()` a `putf()`
  Tieto funkcie slúžia na predávanie obrazových dát zobrazovaciemu modulu. UltraGrid najprv získa „prázdnu“ snímku zo zobrazovacieho modulu, naplní ho obrazovými dátami a potom ho predá naspäť. Toto prúdenie dát je zobrazené na diagrame 6.2.

- `reconfigure()`
  V prípade ak dôjde k zmene formátu vstupného videa je o tom

zobrazovací modul informovaný volaním tejto funkcie. Popis nového formátu je predaný ako parameter.

- `get_property()`
  Pomocou tejto funkcie môže UltraGrid získať informácie o zobrazovacom module, ako napríklad zoznam podporovaných formátov.



Obr. 6.2: Predávanie obrazových dát zobrazovaciemu modulu

6.8.2 Návrh modulu `pano_gl`

Keďže tento zobrazovací modul umožňuje užívateľovi interaktívne ovládať smer pohľadu, je žiadúce, aby tento modul reagoval na užívateľský vstup s čo najmenším oneskorením. Zároveň v prípade ak od užívateľa nedostáva žiaden vstup, by mal v čase kým obdrží novú snímku na zobrazenie využívať čo najmenej zdrojov.

Knižnica SDL využíva na spracovanie vstupu takzvané udalosti. Každé stlačenie klávesy, alebo pohyb kurzoru vytvorí udalosť, ktorá sa zaradí do fronty udalostí. Tieto udalosti je potom možné počas behu zobrazovancieho modulu z tejto fronty vyberať a obsluhovať.

Na efektívne spracovanie udalostí ponúka knižnica SDL funkciu `SDL_WaitEvent()`. V prípade ak je fronta udalostí prázdna, je výpočtové vlákno uspané až do výskytu následujúcej udalosti, čím sa predíde plýtvaniu výpočtových zdrojov v obdobiach „keď sa nič nedeje“. Počas spánku vlákna neprebieha žiaden výpočet, čo ale znamená, že po obdržaní nového snímku na zobrazenie je potrebné toto vlákno nejakým spôsobom prebudiť. Knižnica SDL umožňuje užívateľom definovať vlastné typy udalostí, vďaka čomu je možné definovať typ udalosti pre prichádzajúce snímky. Vložením udalosti tohto typu do fronty udalostí z funckie `putf()` je potom možné vykresľovacie vlákno prebudiť.

6.8.3 Udalosti a ich spracovanie

Udalosti sú v knižnici SDL reprezentované pomocou dátového typu `SDL_Event`. Tento dátový typ je implementovaný ako únia štruktúr reprezentujúcich konkrétne udalosti. Informácia o type udalosti je uložená v položke `type`. Napríklad pri udalosti pohybu kurzora je položka `type` nastavená na `SDL_MOUSEMOTION` a informácie o konkrétnom pohybe sú uložené v štruktúre `motion`, ktorá je v tejto únii obsiahnutá a má typ `SDL_MouseMotionEvent`.

Knižnica SDL umožňuje definovať vlastný typ udalosti. Nový typ udalosti je potrebné pred použitím najprv zaregistrovať pomocou funkcie `SDL_RegisterEvents()`. Táto funkcia vráti celé číslo, ktoré je väčšie alebo rovné hodnote `SDL_USEREVENT`. Pri vytvorení inštancie udalosti je položka `type` nastavená na túto hodnotu. Na uloženie dát udalosti je používaná štruktúra `SDL_UserEvent`, do ktorej je možné uložiť dva ľubovoľné ukazovatele. Napríklad udalosť nového dostupného snímku vytvorená vo funkcii `putf()` využíva jeden z týchto ukazovateľov `data1` na uloženie ukazovateľu na snímku `video_frame`.

Tieto udalosti sú potom v module `pano_gl` spracovávané funkciou `display_panogl_run()`. Táto funkcia obsahuje cyklus, ktorý čaká na udalosti pomocou `SDL_WaitEvent` a následne ich spracovává. Udalosti ako pohyb kurzoru a zmena veľkosti okna ovplyvňujú vykresľovaný obraz a preto je po ich spracovaní vynútené prekreslenie.

6.8.4 Obmedzenie prekresľovania

Niektoré typy udalostí ako pohyb kurzora a zmena veľkosti okna majú tendenciu nastávať veľmi často. Prekresľovanie pri každej udalosti pohybu kurzora by výrazne spomalilo spracovanie a preto je potrebné tieto prekreslenia nejakým spôsobom limitovať.

Prekresľovanie je kontrolované funkciou `redraw()`, ktorá je volaná po spracovaní každej udalosti a rozhoduje či k prekresleniu dôjde. Limitovanie je implementované pomocou časovača, ktorý meria čas od posledného vykreslenia. Ak od vykreslenia ubehlo dostatok času, dôjde k vykresleniu okamžite. V opačnom prípade je nastavený príznak nutnosti prekresliť a kreslenie je odložené.

V knižnici SDL je možné vytvoriť nový časovač pomocou funkcie `SDL_TimerAdd()`. Tieto časovače po uplynutí špecifikovaného časového obdobia zavolajú poskytnutý funkčný ukazovateľ (callback). Tieto časovače sú periodické, čo znamená, že po vypršaní času sa automaticky zresetujú a začnú odpočítavať odznova. Zrušiť opakovanie časovača je možné jeho zničením pomocou funkcie `SDL_RemoveTimer()`.

Funkcia `redraw()` po zavolaní najprv skontroluje, či už časovač existuje. Ak nie, tak ho vytvorí a okamžite prekreslí obraz. V opačnom prípade nastaví príznak `redraw_needed`. Po uplynutí časovača je volaná funkcia ktorá do fronty udalostí pridá užívateľskú udalosť typu `redraw_event`. Pri spracovaní tejto udalosti je v prípade aktívneho príznaku `redraw_needed` vykonané prekreslenie. Ak príznak nastavený nebol, nie je prekreslenie nutné a časovač je zničený, čo zabezpečí, že pri následujúcej udalosti bude možné prekreslenie vykonať okamžite.

6.9 Zhrnutie

V tejto kapitole bol opísaný návrh a implementácia abstrakcie na vykresľovanie 360° videa v ekvidistantnej projekcii pomocou rozhrania OpenGL. Ďalej bolo popísané využitie tejto abstrakcie na implementáciou zobrazovacieho modulu `pano_gl`, ktorý video vykresľuje pomocou tradičného okna na monitore.

7 Implementácia zobrazovania pomocou 3D headsetu

Zobrazovanie 360° videa pomocou headsetu na virtuálnu realitu umožňuje ovládať smer pohľadu jednoduchým otočením hlavy a ponúka výrazne lepší zážitok zo sledovania.

Pre takýto druh zobrazenia bol v programe UltraGrid implementovaný zobrazovací modul s názvom `open_xr`. Na vývoj a testovanie tohto modulu bol k dispozícii len headset Oculus Rift[1] vyrábaný firmou Facebook, ale pri návrhu a implementácii bola snaha podporovať pokiaľ možno čo najviac rôznych headsetov na rôznych platformách.

7.1 Prehľad rozhraní pre komunikáciu s headsetom

Pre prácu s headsetom je potrebné použiť nejaké rozhranie, ktoré umožňuje získavanie informácií potrebných pre korektné vykreslenie snímky a poskytuje nejaký mechanizmus ako vykreslené snímky na headsete zobraziť. Medzi potrebné informácie patria dáta o rotácii headsetu v priestore, umiestnenie a relatívna pozícia displejov pre každé oko, ako aj zorné uhly, ktoré sú ovplyvnené použitými šošovkami. Pri zobrazovaní by malo rozhranie poskytovať mechanizmus ako vykreslený obraz upraviť za účelom korekcie skreslenia spôsobeného šošovkami.

7.1.1 Rozhranie ovládača od výrobcu

Virtuálna realita je pomerne mladá technológia a v období uvedenie prvých modelov headsetov neexistovali štandardné rozhrania. Výrobcovia headsetov preto k svojim headsetom poskytovali vývojové súpravy, ktoré však neboli kompatibilné s headsetmi iných výrobcov. V kontexte headsetu Oculus Rift, ktorý bol použitý na vývoj, to znamená, že použitím takéhoto rozhrania by bola implementácia obmedzená len na headsety Oculus. Vývoj vývojovej súpravy pre platformu Linux, na ktorú sa program UltraGrid sústreďuje, bol ukončený v roku 2015[26].

1. `https://www.oculus.com/rift/`

7.1.2 OpenVR

Na uľahčenie podpory rôznych headsetov vývojárom hier navrhla firma Valve rozhranie OpenVR[2]. Aplikácie využívajúce toto rozhranie potom komunikujú s behovým prostredím (runtime) SteamVR[27]. Toto prostredie je možné používať bezplatne pre nekomerčné účely, no jeho zdrojový kód nie je voľne k dispozícii a na jeho inštaláciu je vyžadovaná registrácia v službe Steam[28].

7.1.3 OpenHMD

Projekt OpenHMD[3] obsahuje open-source implementácie ovládačov pre headsety rôznych výrobcov a vlastné jednotné rozhranie, ktorým je možné s týmito ovládačmi komunikovať[29]. Tieto voľné ovládače fungujú na množstve rôznych platforiem, ale ich kvalita je väčšinou nižšia ako u ovládačov od výrobcu. U väčšiny podporovaných zariadení nie je napríklad podporované sledovanie pozície v priestore. Toto však pri zobrazovaní 360° videa nevadí, pretože to je vždy zobrazované z pozície kamery a možné je meniť len rotáciu pohľadu.

7.1.4 OpenXR

OpenXR je rozhranie navrhnuté skupinou Khronos, ktorá je známa vďaka úspešným štandardom ako OpenGL, Vulkan a OpenCL. Cieľom tohto rozhrania je štandardizovať prácu s rôznymi zariadeniami určenými pre virtuálnu, rozšírenú alebo zmiešanú realitu a tým uľahčiť aplikáciám podporu viacerých zariadení bez nutnosti prepisu častí zdrojového kódu[30]. Aplikácie využívajúce toto rozhranie komunikujú s OpenXR behovým prostredím, ktoré by malo byť súčasťou ovládača od výrobcu. Prvá verzia špecifikácie tohto rozhrania bola vydaná v lete roku 2019, takže sa jedná o pomerne nový štandard, čo znamená, že podpora tohto štandardu headsetmi trochu zaostáva, ale niektorý výrobcovia už vydali ovládače s prvotnou podporou[31][32][33]. Je možné predpokladať, že v budúcnosti bude podpora tohto štandardu veľmi rozšírená, pretože na jeho vývoji sa podieľa väčšina známych

2. `https://github.com/ValveSoftware/openvr`
3. `http://www.openhmd.net/`

výrobcov headsetov a OpenXR je nimi presadzovaný ako preferovaný spôsob vývoja softvéru pre ich hardvér[34].

Na platforme Linux je pre podporu tohto rozhrania k dispozícii open-source behové prostredie Monado[4]. Toto prostredie obsahuje aj otvorené ovládače pre niektoré headsety a je schopné využívať ovládače projektu OpenHMD pre zariadenia ktoré nepodporuje priamo[35].

7.1.5 Zhrnutie

Pre implementáciu zobrazovacieho modulu som vybral rozhranie OpenXR. Aj napriek tomu, že sa jedná o veľmi mladý štandard, javí sa ako najjednoduchší spôsob ako podporovať čo najväčšie množstvo headsetov na čo najväčšom množstve platforiem. Umožňuje využitie kvalitnejších oficiálnych ovládačov na platformách kde sú k dispozícii a súčasne dovoľuje využiť otvorené ovládače na ostatných platformách, bez nutnosti zmien v zdrojovom kóde.

7.2 Inicializácia

Rozhranie OpenXR podporuje veľké množstvo rôzneho hardvéru a umožňuje vykresľovanie pomocou množstva grafických rozhraní. Kvôli týmto vlastnostiam je však pomerne zložité a pred zahájením vykresľovania vyžaduje pomerne rozsiahlu konfiguráciu.

7.2.1 OpenXR Inštancia

Na komunikáciu aplikácie s OpenXR behovým prostredím slúži objekt `XrInstance`[36]. Tento objekt je možné získať pomocou funkcie `xrCreateInstance()`. Parametre tejto funkcie sú predávané pomocou štruktúry `XrCreateInstanceInfo`, ktorá obsahuje niektoré nastavenia a informácie o klientskej aplikácii, ako napríklad jej meno. Zaujímavá je predovšetkým položka `enabledExtensionNames`, ktorá obsahuje pole rozšírení, ktoré by mali byť pre inštanciu aktivované. V tomto prípade sa jedná o rozšírenie `XR_KHR_opengl_enable`, ktoré dovoľuje interoperabilitu rozhraní OpenGL a OpenXR.

4. `https://monado.dev/`

Pred vytvorením inštancie je vhodné skontrolovať, či vyžadované rozšírenia sú na danej platforme k dispozícii. Vykonať túto kontrolu nám umožňuje funkcia `xrEnumerateInstanceExtensionProperties()`, pomocou ktorej môžeme získať zoznam všetkých podporovaných rozšírení.

Na vykonanie tejto kontroly a konfigurácie som implementoval triedu `Openxr_instance`, ktorá zároveň slúži ako RAII obal.

7.2.2 OpenXR systém

Rozhranie OpenXR dovoľuje pripojenie viacerých podporovaných zariadení súčastne. Tieto zariadenia sú potom rozdelené do skupín nazývaných systém, ktoré obsahujú zariadenia, ktoré so sebou nejak súvisia a predpokladá sa, že budú využívané súčasne. Systém môže napríklad obsahovať headset a herné ovládače pre ľavú a pravú ruku.

Tieto systémy sú identifikované pomocou čísla. Pomocou funckie `xrGetSystem()` je možné získať identifikačné číslo pripojeného systému, ktorý spĺňa špecifikované požiadavky. Pre modul `openxr_gl` je použitá požiadavka `XR_FORM_FACTOR_HEAD_MOUNTED_DISPLAY`, ktorá špecifikuje, že požadujeme systém ktorý obsahuje headset obsahujúci displej.

7.2.3 OpenXR sedenie

Konkrétne sedenie aplikácie, ktoré vyjadruje zámer aplikácie vykresľovať snímky je reprezentované dátovým typom `XrSession`. Tento objekt obsahuje väzbu medzi rozhraním OpenXR a grafickým rozhraním využitým na vykresľovanie snímok. Táto väzba je vyjadrená pomocou štruktúry, ktorej typ je závislý od konkrétnej platformy. Na platforme Linux je použitá štruktúra `XrGraphicsBindingOpenGLXlibKHR`, ktorá obsahuje referencie na okno aplikácie a OpenGL kontext s ním zviazaný. Na získanie týchto referencií som v triede `Sdl_window` implementoval funkciu `getXlibHandles()`, ktorá používa volania knižnice Xlib. Pred vytvorením sedenia predaním tejto štruktúry spolu s identifikačným číslom zvoleného systému funkcii `xrCreateSession()` je nutné zavolať funkciu `xrGetOpenGLGraphicsRequirementsKHR()`, ktorá vráti podporované verzie rozhrania OpenGL.

Vytvorené sedenie je možné zahájiť volaním `xrBeginSession()`. Tejto funkcie je potrebné predať konfiguráciu pohľadov ako jeden z parametrov. Pre naše účely stačí zvoliť primárnu stereografickú konfiguráciu `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO`.

Popisované kontroly a inicializácia sedenie je implementovaná triedou `Openxr_session`.

7.2.4 OpenXR priestor

Priestor voči ktorému je sledovaná rotácia a pozícia headsetu je rozhraním OpenXR reprezentovaný pomocou dátového typu `XrSpace`. K dispozícii sú tri referenčné priestory `VIEW`, `LOCAL` a `STAGE`. Objekty v priestore typu `VIEW`, majú súradnice špecifikované relatívne voči pozícii a rotácii headsetu. Tento priestor sa používa na vykresľovanie objektov, ktoré na displeji headsetu nemenia pozíciu pri pohybe alebo rotácii. Priestor `LOCAL` sa používa pre obsah určený na sledovanie v sede. To znamená, že na pozíciu objektov má vplyv rotácia headsetu, no očakáva sa, že užívateľ zostane na mieste. Na zobrazenie obsahu v ktorom sa môže užívateľ prechádzať, teda počíta sa so zmenou pozície headsetu, slúži priestor `STAGE`.

360° video je získané z pozície stredu kamerovej súpravy a preto je pri vykresľovaní zobrazované voči pozícii headsetu staticky. Z tohto dôvodu má zmysel zvoliť priestor typu `LOCAL`. Podpora tohto typu priestoru je pre každé behové prostredie povinná. Získať inštanciu priestoru je možné volaním funkcie `xrCreateReferenceSpace()`.

Podľa špecifikácie rozhrania OpenXR môže byť iniciálna orientácia pohľadu na scénu volená voči pozícii headsetu pri štarte aplikácie, alebo nejakej inej kalibrovanej nulovej pozícii[36].

7.2.5 OpenXR pohľady

Pohľadové konfigurácie spomínané v sekcii 7.2.3 sú zložené z takzvaných pohľadov. Konfigurácia pre typický headset bude obsahovať dva pohľady - jeden pre každé oko, podporované sú však aj zariadenia s viacerými pohľadmi ako napríklad systém zložený s viacerých obrazových stien.

Informácie o jednotlivých pohľadoch zvolenej konfigurácie je možné získať pomocou funkcie `xrEnumerateViewConfigurationViews()`. Jej

volaním je získaná kolekcia štruktúr `XrViewConfigurationView`, ktoré obsahujú napríklad odporúčané rozmery snímky, ktorá by mala byť pre daný pohľad vykreslená.

7.2.6 OpenXR swapchain

Predávanie vykreslených snímok rozhraniu OpenXR prebieha pomocou takzvanej swapchain. Jedná sa o pole niekoľkých snímok, ktoré slúži ako vyrovnávacia pamäť. Pre každý pohľad vytvoríme samostatnú inštanciu swapchain.

Inštanciu typu `XrSwapchain` je možné vytvoriť pomocou funkcie `xrCreateSwapchain()`. Požadované parametre sú funkcii predané pomocou štruktúry `XrSwapchainCreateInfo`, ktorá obsahuje napríklad požadované rozmery snímok a ich formát. Podporované formáty je možné získať volaním funkcie `xrEnumerateSwapchainFormats()`. V module `openxr_gl` je zvolený formát `GL_RGBA8_EXT`, ktorý umožnuje interoperabilitu s rozhraním OpenGL.

Po úspešnom vytvorení swapchain je možné získať pole jednotlivých snímiek pomocou funkcie `xrEnumerateSwapchainImages()`. V prípade ak je formát snímok kompatibilný s rozhraním OpenGL je dátový typ položiek tohto pola `XrSwapchainImageOpenGLKHR`. Položka `image` v tejto štruktúre obsahuje identifikačné číslo OpenGL textúry.

Na vykresľovanie do týchto textúr je použitý OpenGL frame buffer, tak ako bolo opisované v sekcii 6.5. Pre každú snímku v swapchain je vytvorený samostatný buffer.

Na uľahčenie práce som implementoval triedu `Openxr_swapchain`, ktorá sa stará o vytvorenie swapchain a triedu `Gl_interop_swapchain`, ktorá sa stará o vytvorenie a zviazanie frame buffer objektu pre každú snímku zo swapchain. Tieto triedy zároveň fungujú ako RAII obaly, ktoré získané zdroje automaticky uvoľnia po dokončení práce.

7.2.7 Kompozičné vrstvy

Rozhranie OpenXR umožnuje prezentáciu vykresleného obsahu vo viacerých vrstvách. Tieto vrstvy môžu byť použité napríklad na vykreslenie nejakého užívateľského rozhrania alebo menu nad vykreslenú scénu. Do týchto vrstiev je rozdeľovaný všetok obsah určený na vykreslenie a preto je nutné vytvoriť aspoň jednu vrstvu.

Rozhranie OpenXR definuje viac druhov vrstiev, no pre naše účely je vhodný typ `XrCompositionLayerProjection`. Tento typ reprezentuje obsah vykreslený z pozície každého oka pomocou perspektívnej projekcie. Definícia tejto vrstvy obsahuje referenciu na použitý OpenXR priestor a pole štruktúr `XrCompositionLayerProjectionView` popisujúcich pohľady patriace do tejto vrstvy. Pre každý pohľad je uchovaná referencia na swapchain, ktorý je viazaný k danému pohľadu a rozmery textúry.

7.3 Vykresľovací cyklus

Po inicializácii je možné zahájiť vykresľovanie snímok. Vykresľovanie je vykonávané sekvenciou krokov, ktoré sa opakujú pre každú snímku a sú implementované pomocou cyklu vo funkcii `display_xrgl_run()`.

7.3.1 Časovanie snímok

Pri zobrazovaní obsahu pomocou headsetu pre virtuálnu realitu je veľmi dôležité, aby bol pohyb čo najplynulejší, pretože rôzne trhania môžu u užívateľa vyvolávať pocity nevoľnosti. Rozhranie OpenXR preto ponúka mechanizmus, ktorým je možné dosiahnuť presné časovanie snímok. Slúži na to funkcia `xrWaitFrame()`. Táto funkcia je volaná na začiatku každej iterácie cyklu. Volaním je výpočtové vlákno uspané a následne prebudené v čase vhodnom na kreslenie. Pri prebudení je vyplnený obsah štruktúry `XrFrameState`, ktorá obsahuje predvídaný čas zobrazenia vykresľovanej snímky.

7.3.2 Získanie rotácie pohľadu

Aby bolo možné vykresliť snímky v správnej orientácii je potrebné poznať rotáciu headsetu v čase zobrazenia snímky. Na získanie tejto informácie je v rozhraní OpenXR k dispozícia funckia `xrLocateViews()`, ktorá vráti pozíciu, rotáciu a zorné uhly pre každý pohľad. Medzi parametre tejto funkcie je aj predpokladaný čas zobrazenia snímky a preto môže vykonávať predikciu týchto dát na základe zotrvačnosti.

Pomocou získaných dát je možné vytvoriť transformačnú maticu zloženú z projekčnej a pohľadovej matice podobne ako pri module

`pano_gl` v sekcii 6.6.1. Reprezentácia dát získaných z rozhrania OpenXR je však odlišná a preto je konštrukcia častí tejto matice mierne odlišná.

Pre konštrukcie matice perspektívnej projekcie som v prípade modulu `pano_gl` použil funkciu `glm::perspective`. Táto funkcia konštruovala maticu na základe horizontálneho zorného uhlu. Rozhranie OpenXR však poskytuje túto informáciu pomocou štruktúry `XrFovf`, ktorá obsahuje štyri separátne zorné uhly v smere nahor, nadol, vľavo a vpravo. To znamená, že zorný uhol nemusí byť izotropný. Knižnica `glm` neobsahuje funkciu ktorej je možné špecifikovať horizontálny a vertikálny uhol separátne, preto bolo nutné implementovať vlastnú funkciu na konštrukciu tejto matice. Výpočet matice je vykonaný podľa známych vzorcov[37].

Rotácia headsetu je rozhraním OpenXR reprezentovaná pomocou kvaterniónu. Konverziu kvaterniónu na rotačnú maticu je možné vykonať pomocou knižnice `glm`, ktorá ponúka konverzie medzi typmi `glm::quat` a `glm::mat4`. Táto rotačná matica však reprezentuje rotáciu samotného headsetu. Aby ju bolo možné požiť na konštrukciu transformačnej matice aplikovateľnej na vykresľované objekty je potrebné získať jej inverznú maticu, ktorú je možné získať pomocou funkcie `glm::inverse()`.

V niektorých prípadoch nemusí iniciálna orientácia pohľadu na scénu vyhovovať, napríklad ak zvislý smer v zobrazovanom videu neodpovedá zvislému smeru v reálnom živote, čo spôsobí, že sa zobrazený obsah javí ako „naklonený“. V takýchto situáciách je žiadúce resetovať smer považovaný za nulovú rotáciu. Toto je možné vykonať uložením aktuálnej získanej rotácie. Táto uložená rotácia je potom použitá pri výpočte transformačnej matice tak, aby bola scéna zrotovaná do správnej pozície vzhľadom k pozorovateľovi.

7.3.3 Kreslenie snímky

Rozhranie OpenXR vyžaduje, aby pred akýmkoľvek kreslením bola snímka „zahájená“ volaním funkcie `xrBeginFrame()`. Po zahájení nasleduje kreslenie každého pohľadu. Prvým krokom je získanie cieľovej textúry do ktorej sa bude vykresľovať zo swapchain daného pohľadu. Toto je vykonané pomocou `xrAcquireSwapchainImage()`. Pred samotným kreslením je ešte nutné vykonať synchronizáciu, aby sa predišlo

prepisu dát, ktoré sú ešte čítané OpenXR behovým prostredím. Táto synchronizácia je vykonaná volaním `xrWaitSwapchainImage()`.

Po získaní textúry je aktivovaný viazaný framebuffer a scéna je vykreslená pomocou funkcie `render()` objektu `Scene`. Aby bolo možné využiť transformačnú maticu, ktorá bola vytvorená tak ako bolo opísané v predchádzajúcej sekcii je implementovaná varianta tejto funkcie, ktorej je možné túto maticu predať ako parameter.

Po vykreslení jedného z pohľadov je možné tento pohľad vykresliť aj do okna na obrazovke, ktoré slúži ako náhľad. Toto je vykonané pomocou funkcie `glBlitNamedFramebuffer()`, ktorá umožňuje kopírovanie obsahu medzi frame buffer inštanciami.

Získanú textúru zo swapchain je po dokončení kreslenia potrebné uvoľniť rozhraniu OpenXR. Keďže OpenGL vykonáva pokyny na kreslenie asynchrónne, je pred takýmto uvoľnením potrebné zaručiť, že kreslenie sa už skutočne dokončilo. Toto je možné vykonať pomocou funkcie `glFinish()`, ktorá blokuje ďalšie výpočty, dokiaľ nie je vykonaná všetka práca v OpenGL fronte. Následne je bezpečné textúru uvoľniť pomocou funkcie `xrReleaseSwapchainImage()`.

Na záver je po vykreslení všetkých pohľadov potrebné ukončiť zahájenú snímku pomocou funkcie `xrEndFrame()`. Parametre sú tejto funkcii predané pomocou štruktúry `XrFrameEndInfo`, do ktorej je zapísané pole kompozičných vrstiev a nastavenie miešania medzi vrstvami.

7.4 Optimalizácie

Na zaistenie výkonu potrebného k dosiahnutiu plynulosti, nutnej na zamedzenie pocitov nevoľnosti boli implementované následujúce optimalizácie.

7.4.1 Pamäťové prenosy s využitím Pixel buffer objektov

Pri nahrávaní obsahu textúr do pamäte grafickej karty štandardným spôsobom, sú tieto dáta ovládačom najprv kopírované do dočasnej vyrovnávacej pamäte, ktorá je alokovaná špeciálnym spôsobom, ktorý umožňuje vykonať asynchrónny DMA (Direct Memory Access) prenos. PBO (Pixel Buffer Object) je nástroj, ktorým OpenGL umožňuje

takúto pamäťovú oblasť alokovať manuálne a tým sa vyhnúť zbytočnému kopírovaniu[38].

Prvým krokom takejto alokácie je získanie nového buffer objektu volaním funkcie `glGenBuffers()`. Následne je tento buffer aktivovaný ako `GL_PIXEL_UNPACK_BUFFER`, teda ako buffer určený na nahrávanie obrazových dát do grafickej karty. Po aktivácii je nastavená veľkosť tohto bufferu volaním `glBufferData()` s nulovým ukazovateľom a požadovanou veľkosťou. Ukazovateľ pomocou ktorého do bufferu môžeme zapisovať získame volaním `glMapBuffer()`. Po zápise obrazových dát je mapovanie bufferu zrušené volaním `glUnmapBuffer` a nahrané do textúry pomocou `glTexSubImage2D()`. Táto funkcia počas doby aktivácie bufferu typu `GL_PIXEL_UNPACK_BUFFER` interpretuje parameter `data` ako offset začiatku zdrojových dát od začiatku aktívneho bufferu.

Keďže zobrazované video snímky sú v programe UltraGrid alokované zobrazovacím modulom, je možné takto získanú pamäťovú oblasť predať kódu zodpovednému za prijímanie a dekódovanie videa.

Na uloženie referencie OpenGL bufferu, do ktorého obsahuje snímka `video_frame` ukazovateľ je použitá položka `dispose_udata` tejto štruktúry, ktorá sa v programe UltraGrid používa na uloženie ukazovateľa na prípadné extra dáta. Zároveň je nastavenie tohto ukazovateľa možno použiť ako príznak, že sa jedná o snímku využívajúci Pixel buffer objekt namiesto klasickej alokácie pamäte.

Na organizáciu snímok je použitá fronta vstupných snímok, pole pre voľné snímky a pole snímok, ktoré už boli vykreslené a môžu sa recyklovať na voľné snímky. Pri spracovaní snímok určených na recykláciu sa najprv skontroluje, či ich formát obrazu je zhodný s požadovaným formátom voľných snímok. Ak nie, tak je snímka zničená a namiesto nej je vytvorená nová. Ak sa formát zhoduje, tak je volaná funkcia `recycle_frame()`. Táto funkcia najprv uvoľní mapovanie bufferu, následne vynuluje jeho obsah volaním `glBufferData()` s nulovým ukazovateľom a potom buffer opäť namapuje.

Po dokončení práce so zobrazovacím modulom sa všetky snímky zničia pomocou funkcie `delete_frame()`, ktorá sa stará o korektné zrušenie mapovania a uvoľnenie bufferu.

7.4.2 Nahrávanie video snímok z oddeleného vlákna

Frekvencia vykresľovania snímok je určená obnovovacou frekvenciou headsetu, aby bola dosiahnutá čo najkratšia reakcia na zmenu rotácie headsetu. V prípade, že snímková frekvencia videa je nižšia, nedochádza k nahrávaniu novej video snímky do textúry pri každom vykresľovaní. Výsledkom je, že kreslenie snímok pri ktorých dochádza k nahrávaniu novej video snímky do textúry trvá dlhší čas. Táto variácia časov kreslenia má však negatívny dopad na predikciu rotácie headsetu a môže vyvolávať nepríjemné trhanie.

Na vyriešenie tohto problému som implementoval nahrávanie video snímok z oddeleného vlákna, ktoré používa vlastný OpenGL kontext a tým pádom aj vlastnú OpenGL frontu práce.

Knižnica SDL umožnuje vytvorenie OpenGL kontextov, ktoré sú medzi sebou schopné zdielať niektoré zdroje ako napríklad textúry pomocou nastavenie atribútu `SDL_GL_SHARE_WITH_CURRENT_CONTEXT`[39]. Pre zachovanie korektnosti pri využití takéhoto zdieľania je nutné prístup k zdielaným prostriedkom serializovať, aby sa predišlo prepisovaniu práve čítaných dát. Trieda `Scene` bola implementovaná takým spôsobom, aby funkcie `put_frame()` a `render()` mohli byť volané z oddelených vlákien a vykonávali potrebnú synchronizáciu.

Pretože vertex array objekty, ktoré sú používané pri vykresľovaní a konverzii formátu obrazových bodov nie je možné zdielať medzi kontextmi[40], musia byť tieto objekty vytvorené a inicializované vláknami, ktoré ich budú používať. Z tohto dôvodu je objekt `Yuv_convertor` vytvorený až za behu funkcie `put_frame()`. Táto funkcia môže byť následne volaná len z vlákna, ktoré ju volalo ako prvé.

Po nahraní snímky a jej prípadnej konverzii je pred sprístupnením tejto textúry vykresľovaciemu vláknu nutné uistiť sa, že jej zápis je už dokončený. Toto je vykonané pomocou funkcie `glFinish()` tak ako to bolo opísaná v sekcii 7.3.3. Na samotné predávanie snímok medzi vláknami je použitá technika s názvom triple buffering. Jedná sa o vyrovnávaciu pamäť zloženú z troch textúr - prednej, ktorá obsahuje textúru určenú na vykreslenie, zadnej, ktorá slúži na zápis nových dát a strednej, ktorá slúži na ich vymieňanie. Pri použití tejto techniky nemusí vlákno pri synchronizácii čakať, kým druhé vlákno dokončí prácu, pretože každé vlákno má vždy k dispozícii svoju snímku s ktorou môže konať prácu.

Na implementáciu tejto techniky výmeny textúr som vytvoril triedu `TripleBufferTexture`. Obsahuje spomenuté tri textúry spolu s príznakom `new_front_available`, ktorý signalizuje, či je k dispozícii nová textúra na čítanie. Vlákno po nahratí nových dát do zadnej textúry túto textúru vymení so strednou pomocou funkcie `swap_back()`, ktorá súčasne nastaví príznak. Vlákno, ktoré vykresľuje, číta textúru prednú. V prípade ak je príznak novej textúry nastavený, vymení strednú a prednú textúru pomocou `swap_front()`, ktorá súčasne príznak resetuje. Tieto výmeny sú veľmi rýchle a preto dochádza k zamykaniu synchronizačného zámku len na veľmi krátke doby.

7.5 Zhrnutie

V tejto kapitole boli opísané návrhové rozhodnutia a implementácia zobrazovacieho modulu `openxr_gl`, ktorý slúži na zobrazovanie 360° videa pomocou headsetu na virtuálnu realitu. Po implementácii opísaných optimalizácií tento modul ponúka dostatočný výkon pre plynulé zobrazenie.

8 Meranie výkonu

Táto kapitola sa zoberá meraním výkonu a oneskorenia skladania 360° videa ako aj jeho zobrazovania. Na meranie výkonu a oneskorenia je využitá inštrumentácia využívajúca zaznamenávanie časov z monotónnych hodín.

8.1 Časovač na inštrumentáciu

Pre uľahčenie inštrumentácie kódu bežiaceho na procesore som implementoval súbor tried, pomocou ktorých je možné jednoducho vytvoriť časovač, ktorý meria čas strávený vo funkcii a výsledky zapisuje do súboru na disku. Výstupný súbor je kompatibilný s profilovacím nástrojom zabudovaným v prehliadači Google Chrome[1], ktorým je možné získané dáta vizualizovať. Implementácia týchto tried a nachádza v súbore `src/utils/profile_timer.cpp`.

Idea tohto riešenia je, že na začiatku funkcií, ktoré chceme merať, je skonštruovaná inštancia triedy `Profile_timer`. Pri vytvorení tejto inštancie dôjde k zaznamenaniu aktuálneho času pomocou monotónnych hodín `std::chrono::steady_clock`. Tieto hodiny sú definované ako neklesajúce a nemali by byť ovplyvnené zmenami nastavenia času[41]. Po skončení behu meranej funkcie skončí životnosť tohto objektu a pri jeho deštrukcii je opäť zaznamenaný aktuálny čas. Takto získaný čas začiatku a konca behu funkcie je potom uložený do vyrovnávacej pamäte a po akumulácii väčšieho počtu takýchto časových úsekov sú naraz zapísané na disk.

Na vytváranie týchto inštancií časovačov je použité makro preprocesora, ktoré automaticky poskytne konštruktoru názov funkcie z ktorej je použité. Meranie výkonu a zápis týchto výsledkov na disk je vďaka použitiu makier ľahko aktivovateľné počas prekladu bez nutnosti modifikácie zdrojového kódu. V prípade ak je meranie deaktivované sú makrá definované s prázdnym telom a žiaden inštrumentačný kód sa vôbec nepreloží.

---

1. `http://dev.chromium.org/developers/how-tos/trace-event-profiling-tool`

Aby bolo možné tieto inštrumentačné udalosti generovať súčasne z viacerých vlákien je potrebné zabezpečiť serializovaný prístup k výstupnému súboru. Pre minimalizáciu vplyvu synchronizácie používa každé vlákno na uskladnenie udalostí vlastnú vyrovnávaciu pamäť.

8.2 Použité počítače

Na merania boli použité počítače **HD13** a **AMD1**. Aby bolo možné merania vykonávať na vstupných dát z reálnych kamier, obsahuje počítač **HD13** kartu Decklink 8K Pro, ktorá slúži ako SDI rozhranie pre kamery. Prítomnosť tejto karty v spojení s obmedzeným počtom PCIe liniek na použitej základnej doske však spôsobuje, že grafická karta na komunikáciu využíva len 8 PCIe liniek namiesto podporovaných 16. Toto obmedzenie negatívne ovplyvňuje rýchlosť pamäťových procesov medzi grafickou kartou a CPU pamäťou.

Na počítači **AMD1** boli merania vykonávané len s použitím modulu `testcard`, ktorý slúži na generovanie testovacieho obrazu. Tento modul je zároveň používaný aj na testovanie chovania výslednej implementácie pri vyšších snímkových frekvenciách ako sú podporované kamerovou súpravou, ktorá bola k dispozícii.

HD13

Operačný systém: Ubuntu 18.04.4

Základná doska: ASUS Z170-A

CPU: Intel® Core™ i7-6700K

Operačná pamäť: 4*4 GB 3333 MHz DDR4

GPU: NVIDIA GeForce GTX 1080 Ti (8 PCIe liniek)

Playback/capture karta: DeckLink 8K Pro

AMD1

Operačný systém: Arch Linux

Základná doska: MSI MEG x570 ACE

CPU: AMD Ryzen™ 7 3700X

Operačná pamäť: 2*8 GB 3000 MHz DDR4

GPU: NVIDIA GeForce RTX 2070

8.3 Meranie skladania obrazu

Na meranie výkonu a oneskorenia skladania obrazu bol použitý nástroj opisovaný v sekcii 8.1. Zaznamenávané boli časy behu funkcie `vidcap_gpustitch_grab()`, ktorou modul `gpustitch` predáva výsledné snímky programu UltraGrid na ďalšie spracovanie a funkcie `grab()` zaznamenávacieho modulu, z ktorého sú získavané vstupné snímky. Pre odmeranie oneskorenia sledujeme časový rozdiel od získania vstupných snímok po odovzdanie výslednej snímky. Pre merania celkovej priepustnosti sledujeme časový rozdiel medzi odovzdaním po sebe idúcich výstupných snímok.

Pri spracovaní nameraných dát sa najprv zahodili časy prvých sto snímok, aby sa z dát odstránili prípadné abnormálne hodnoty namerané pred ustálením snímkovej frekvencie. Následne boli vybrané časy následujúcich sto snímok a z nich boli vypočítané priemery a smerodajné odchýlky.

8.3.1 Testovanie očakávaného použitia

Pri prvej sade testov boli testované prípady, o ktorých sa predpokladá, že budú najčastejšie využívané pri reálnom použití. Vstupné aj výstupné video je v rozlíšení 8K pri použití formátu obrazových bodov UYVY. Ako blending algoritmus bol použitý multiband blending s Laplacovými pyramídami obsahujúcimi štyri levely. Aby sa zamedzilo ovlplyvnenie merania nežiadúcimi faktormi ako výkon sieťovej vrstvy, bol pri meraní v programe UltraGrid využitý transportný

protokol `loopback`, ktorý namiesto sieťového transportu snímky len kopíruje a testovací zobrazovací modul `dummy`, ktorý snímky namiesto zobrazenia len zahadzuje. Namerané hodnoty je možné vidieť v tabuľke 8.1.

Z nameraných hodnôt je vidieť, že oneskorenia a jeho stabilita sa zhoršujú so zvyšujúcou sa záťažou. Najvyšší výkon bol nameraný na počítači **AMD1**, ktorý dosiahol priemernú periódu medzi po sebe idúcimi snímkami 15.06 ms (65.96 snímok za sekundu) aj napriek tomu, že použitá grafická karta obsahuje menej výpočtových CUDA jadier[42][43]. Pri analýze behu s využitím nástroja nvprof[2] bolo možné pozorovať, že tento rozdiel je spôsobený pomalšími pamäťovými prenosmi spôsobenými redukovaným počtom využitých PCIe liniek. V tabuľke 8.2 je možné vidieť namerané hodnoty pamäťových prenosov. Prenos hotového snímku z grafickej karty je rýchlejší a stabilnejší, pretože využíva DMA prenos do uzamknutej pamäte. Prenos vstupných snímok do grafickej karty neprebieha z uzamknutej pamäte a preto nie je možné použiť DMA prenos. Prenos takýmto spôsobom nezaťaží 16 PCIe liniek plne a preto je rýchlosť prenosu menej stabilná.

Tabuľka 8.1: Namerané hodnoty oneskorenia. Použitý blending algoritmus bol multiband so štyrmi levelmi. Formát vstupného aj výstupného videa je 8K UYVY.

Počítač	Počet vstupných snímok za sekundu	Oneskorenie priemer [ms]	Smerodajná odchýlka oneskorenia [ms]	Priemerná perióda snímok [ms]	Smerodajná odchýlka periódy [ms]
AMD1	90	21.53	0.48	15.06 (66.4 fps)	0.52
AMD1	60	19.42	0.20	16.67 (59.99 fps)	0.25
AMD1	30	15.83	0.13	33.34 (29.99 fps)	0.17
HD13	60	29.31	0.06	17.01 (58.79 fps)	0.04
HD13	30	23.80	0.04	33.33 (30.00 fps)	0.05

2. `https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview`

Tabuľka 8.2: Namerané doby pamäťových prenosov

Počítač	Smer prenosu	Priemerná doba [ms]	Smerodajná odchýlka [ms]
AMD1	do GPU	1.56	0.12
AMD1	z GPU	4.36	0.0001
HD13	do GPU	2.58	0.0044
HD13	z GPU	8.95	7.15×10^{-5}

8.4 Merania zobrazovacieho modulu `openxr_gl`

Merania zobrazovacieho modulu `openxr_gl` prebiehali na počítači **AMD1** s headsetom Oculus Rift. Ako behové prostredie bolo použité prostredie Monado vo verzii `r2216.d3ccbce7-1` spolu s ovládačmi OpenHMD verzie `0.3.0.397.2c2dccc-1`.

OpenGL funkcie bývajú tradične spúšťané na grafickej karte asynchrónne a preto sa na meranie dĺžky ich behu nedá použiť časovač zaznamenávajúci rozdiel časov pred volaním funkcie a po jej vrátení. Týmto spôsobom by sa totiž odmeral len čas potrebný na zaradenie danej operácie do fronty práce. Modul `openxr_gl` však na synchronizáciu používa volania funkcie `glFinish()`, ktorá blokuje dokým sa všetka práca vo fronte nedokončí. Toto nám umožňuje odmerať celkovú dobu potrebnú na naplánovanie a dokončenie práce pre každú snímku.

Ďalej bolo sledované vykresľovanie jednotlivých pohľadov pomocou Frekvencia vykresľovania snímok do headsetu je nastavená podľa obnovovacej frekvencie použitého headsetu.

8.4.1 Meranie výkonu nahrávania videosnímok do textúry

Pri meraniach bol sledovaný čas behu funkcie `put_frame` triedy `Scene`, ktorá beží v samostatnom vlákne a zodpovedá za nahranie vstupnej snímky do textúry. Z nameraných dát bol vypočítaný priemerný čas behu tejto funkcie a smerodajná odchýlka. Ďalej bol vypočítaný čas medzi dobehnutím tejto funkcie po sebe idúcich snímok, ktorý by mal odpovedať snímkovej frekvencii vstupného videa. Sledovaním tejto ve-

ličiny pri použití vstupného videa s veľkou snímkovou frekvenciou je možné zistiť celkovú priepustnosť. Namerané hodnoty sú k dispozícii v tabuľke 8.3.

Tabuľka 8.3: Namerané hodnoty doby behu nahrávania snímky

Formát videa	Priemerná doba behu [ms]	Smerodajná odchýlka doby behu [ms]	Priemerná perióda [ms]	Smerodajná odchýlka periódy [ms]
4K UYVY 60fps	1.55	0.09	16.67 (59.99 fps)	0.17
4K UYVY 240fps	1.50	0.17	4.16 (240.39 fps)	0.26
4K UYVY 500fps	1.39	0.08	1.98 (505.05 fps)	0.15
8K RGB 30fps	7.03	0.01	33.34 (29.99 fps)	0.05
8K UYVY 30fps	5.50	0.07	33.33 (30.00 fps)	0.13
8K RGB 60fps	7.02	0.07	16.67 (59.99 fps)	0.14
8K UYVY 60fps	5.39	0.03	16.67 (59.99 fps)	0.17
8K RGB 240fps	6.90	0.08	6.90 (144.93 fps)	0.09
8K UYVY 240fps	5.13	0.13	5.14 (194.55 fps)	0.13

Z nameraných údajov je vidieť, že nahrávanie vstupných videosnímok do textúry má pomerne stabilnú dobu behu. Výsledná implementácia ponúka zároveň dostatočný výkon pre spracovanie $\frac{1s}{6.90ms} = 144$ snímok za sekundu v prípade 8K RGB videa a $\frac{1s}{5.14ms} = 194$ snímok za sekundu v prípade 8K UYVY videa. Pri formáte obrazových bodov UYVY musí dochádzať ku konverzii a preto sa môže zdať vyšší výkon pri tomto formáte prekvapivý. Video v tomto formáte je však reprezentované menším množstvom dát a preto je pamäťový prenos do grafickej karty rýchlejší.

8.4.2 Meranie výkonu a oneskorenie vykreslenia snímok

Pre meranie výkonu vykresľovania snímok bol sledovaný čas pred zavolaním funkcie `render()` triedy `Scene` a čas dokončenia funkcie

`glFinish()`, ktorá je po tejto funkcii volaná. Zároveň je meraný čas medzi dobehnutím funkcie `xrEndFrame()` po sebe následujúcich snímok. Keďže kreslenie a zobrazovanie snímok headsetom je viazané na obnovovaciu frekvenciu použitého headsetu, by táto hodnota mala byť stabilná. V prípade headsetu Oculus Rift je obnovovacia frekvencia 90 snímok za sekundu a preto sa očakáva, že nameraná hodnota periódy sa bude pohybovať okolo $\frac{1s}{90Hz} = 11.11ms$. Vo všetkých testovacích prípadoch sa táto hodnota správala podľa tohto očakávania a preto namerané hodnoty v tabuľke 8.4 obsahujú len smerodajnú odchýlku tejto veličiny.

Tabuľka 8.4: Namerané hodnoty výkonu kreslenia

Formát videa	Priemerná doba kreslenia [ms]	Smerodajná odchýlka doby kreslenia [ms]	Smerodajná odchýlka periódy [ms]
4K UYVY 60fps	0.13	0.05	0.17
4K UYVY 240fps	0.20	0.19	0.18
4K UYVY 500fps	0.18	0.15	0.14
8K RGB 30fps	0.14	0.03	0.07
8K UYVY 30fps	0.23	0.21	0.50
8K RGB 60fps	0.13	0.03	0.17
8K UYVY 60fps	0.15	0.04	0.13
8K RGB 240fps	0.16	0.01	0.12
8K UYVY 240fps	0.20	0.09	0.18

Z nameraných údajov je možné pozorovať, že vykresľovanie a predávanie snímok je pomerne stabilné vo všetkých testovacích prípadoch. Toto kreslenie prebieha v samostatnom vlákne a ani pri plnej záťaži vlákna na nahrávanie vstupných videosnímok nedochádza k žiadnemu výraznému spomaleniu. V prípadoch keď malo vstupné video formát obrazových bodov UYVY bola doba kreslenia a smerodajná

odchýlka mierne vyššia. Toto je pravdepodobne spôsobené tým, že pri konverzii je vykonávané vykresľovanie do textúry.

Oneskorenie, teda čas od získania vstupnej videosnímky po jej zobrazenie, je ovplyvnený viacerými faktormi. Keďže kreslenie beží kompletne nezávisle od vstupu videosnímok, je okrem doby nahrávania vstupnej videosnímky a doby kreslenia potrebné uvážiť aj obnovovaciu frekvenciu použitého headsetu. V prípade, keď je nová videosnímka k dispozícii tesne po vykreslení a zobrazení predošlej, je nutné počkať celú periódu pred tým ako môže byť vykreslená. Oneskorenie je potom možné vypočítať ako súčet doby nahrávania, periódy obnovovacej frekvencie headsetu a doby vykreslenia. Pri výpočte je potrebné uvážiť, že kreslenie pre každú snímku prebieha dvakrát (jedenkrát pre každé oko). Pri uvážení priemernej doby čakania na zahájenie kreslenia snímky $5.55ms$ je pri testovacom prípade 8K UVYY 60fps oneskorenie $5.39ms + 5.55ms + 2 \times 0.15ms = 11.24ms$.

8.5 Merania zobrazovacieho modulu `pano_gl`

Merania modulu `pano_gl` prebiehali na počítači **AMD1**. Tento modul po kreslení normálne nevykonáva synchronizáciu pomocou `glFinish()`. Aby bolo možné korektne získať čas dokončenia kreslenie bolo toto volanie za účelom merania do kódu dočasne doplnené. Keďže samotné kreslenie je vykonávané triedou `Scene` a už bolo premerané pri meraniach modulu `openxr_gl`, budú sa merania v tejto sekcii sústreďovať na celkový výkon a oneskorenie modulu `pano_gl`. Meraný bol čas spracovania snímky od začiatku jej nahrávania do textúry po vykreslenie výsledku pomocou volania `SDL_GL_SwapWindow()` a perióda vykresľovaných snímok. Výsledky sú v tabuľke 8.5.

Z nameraných hodnôt je vidieť, že modul `pano_gl` má oproti modulu `open_xr` menší výkon. Dôvodom je, že tento modul neobsahuje optimalizácie nahrávania textúr pomocou PBO tak ako bolo opísané v kapitole 7.4. Nahrávanie snímky do textúry preto vykonáva kopírovanie dát naviac.

Tabuľka 8.5: Namerané hodnoty výkonu a oneskorenia

Formát videa	Priemerné oneskorenie [ms]	Smerodajná odchýlka oneskorenia [ms]	Priemerná perióda [ms]	Smerodajná odchýlka periódy [ms]
8K UYVY 30fps	10.85	0.57	33.34 (29.99 fps)	0.97
8K UYVY 60fps	10.91	0.66	16.67 (59.99 fps)	0.90
8K UYVY 200fps	12.84	0.55	12.91 (77.46 fps)	0.55

8.6 Zhrnutie

Táto kapitola obsahuje popis a výsledky merania oneskorenia a výkonu výslednej implementácie skladania 360° obrazu a jeho zobrazovania.

Merania ukázali, že implementované skladanie videa dosahuje na dostupnom hardvéri dostatočný výkon na skladanie videa v rozlíšení 8K so snímkovou frekvenciou 60 snímok za sekundu v reálnom čase. Skladanie videa do systému vnáša mierne oneskorenie, ktoré ale nie je výrazné.

Namerané hodnoty pre implementáciu zobrazovania ukázali, že výsledná implementácia má dostatočný výkon na vykresľovanie videa v rozlíšení 8K aj pri snímkových frekvenciách vyšších ako sa bežne používajú.

9 Záver

Cieľom tejto práce bola implementácia skladania 360° videa a jeho zobrazovanie pre nástroj UltraGrid. Implementácia skladania obrazu využíva platformu CUDA za účelom akcelerácie na grafických kartách.

Zobrazovanie videa je umožnené pomocou headsetu pre virtuálnu realitu ako aj tradične v okne na monitore. Implementácia zobrazovania pomocou headsetu využíva rozhranie OpenXR, vďaka čomu podporuje veľké množstvo headsetov rôznych výrobcov.

Po dokončení implementácie boli vykonané merania výkonu a oneskorenia, ktoré je do nástroja UltraGrid vnesené spracovaním obrazu. Meranie výkonu implementácie skladania ukázalo, že poskytujedostatočný výkon na skladanie videa v rozlíšení 8K pri snímkovej frekvencii 60 snímok za sekundu v reálnom čase. Merania zobrazovania ukázali, že implementácia má dostatočný výkon na vykresľovanie videa aj pri snímkových frekvenciách vyšších ako sa bežne používajú.

Zdrojový kód programu UltraGrid s vykonanými zmenami je k dispozícii na `https://github.com/mpiatka/UltraGrid` vo vývojových vetvách `openxr` a `gpustitch`. Zdrojový kód knižnice na skladanie videa je k dispozícii na `https://github.com/mpiatka/gpustitch`. Zdrojový kód je taktiež súčasťou elektronickej prílohy práce.

Výsledná implementácia ponúka priestor pre ďalšie rozšírenia. Knižnica na skladanie 360° videa by mohla byť rozšírená podporou pre kamerové súpravy využívajúce odlišné rozloženia ako prstencové. Ďalej by mohla byť implementovaná podpora pre dodatočné projekcie 360° videa, ktoré by mohli poskytovať lepšie vlastnosti v spojení so stratovou kompresiou videa[44]. Implementácia zobrazovania pomocou headsetu by mohla byť rozšírená o platformovo závislé väzby medzi rozhraniami OpenXR a OpenGL, čo umožní implementáciu využívať na týchto platformách.

Bibliografia

1. MILLER, Benjamin. *With video display's approaching 8K, is there a limit to how sharp an image can be before the user will no longer notice improvement?* Dostupné tiež z: https://eu.mouser.com/applications/display-technology/.
2. HIRSCH, Robert. *Light and Lens: Photography in the Digital Age*. 2. vyd. Focal Press, 2012. ISBN 978-0-240-81827-6.
3. *Terms you need to know to create 360 video* [online]. Vimeo, 2017 [cit. 2020-08-22]. Dostupné z : https://vimeo.com/blog/post/terms-you-need-to-know-to-create-360-video/.
4. BINDER, Daniel. *Re: URSA mini 4K EF suitable for live productions? (DELAY?)* [Online]. 2016 [cit. 2020-12-31]. Dostupné z : https://forum.blackmagicdesign.com/viewtopic.php?f=2&t=48645#p282195.
5. *cv::Stitcher Class Reference* [online]. 2020 [cit. 2020-08-30]. Dostupné z : https://docs.opencv.org/master/d2/d8d/classcv_1_1Stitcher.html.
6. *Introducing Facebook Surround 360: An open, high-quality 3D-360 video capture system* [online]. Facebook, 2016 [cit. 2020-08-30]. Dostupné z : https://engineering.fb.com/video-engineering/introducing-facebook-surround-360-an-open-high-quality-3d-360-video-capture-system/.
7. *VRWorks - 360 Video* [online]. NVIDIA [cit. 2019-10-04]. Dostupné z : https://developer.nvidia.com/vrworks/vrworks-360video.
8. FANNIN, T.E.; GROSVENOR, T. *Clinical Optics*. Elsevier Science, 2013. ISBN 9781483192598. Dostupné tiež z: https://books.google.cz/books?id=IbQ3BQAAQBAJ.
9. *Lens correction model* [online] [cit. 2020-09-20]. Dostupné z : https://wiki.panotools.org/Lens_correction_model.
10. RUBLEE, Ethan; RABAUD, Vincent; KONOLIGE, Kurt; BRADSKI, Gary. ORB: an efficient alternative to SIFT or SURF. In: 2011, s. 2564–2571. Dostupné z DOI: 10.1109/ICCV.2011.6126544.

11. ROSTEN, E.; DRUMMOND, T. Fusing points and lines for high performance tracking. In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. 2005, zv. 2, 1508–1515 Vol. 2. Dostupné z DOI: 10.1109/ICCV.2005.104.

12. BRUNOPOSTLE. *Panorama Tools optimize.c* [online]. 2003 [cit. 2020-12-30]. Dostupné z : https://sourceforge.net/p/panotools/libpano13/ci/default/tree/optimize.c.

13. ZHU, Z.; LU, J.; WANG, M.; ZHANG, S.; MARTIN, R. R.; LIU, H.; HU, S. A Comparative Study of Algorithms for Realtime Panoramic Video Blending. *IEEE Transactions on Image Processing*. 2018, roč. 27, č. 6, s. 2952–2965. Dostupné z DOI: 10.1109/TIP.2018.2808766.

14. BURT, Peter J.; ADELSON, Edward H. A Multiresolution Spline with Application to Image Mosaics. *ACM Trans. Graph.* 1983, roč. 2, č. 4, s. 217–236. ISSN 0730-0301. Dostupné z DOI: 10.1145/245.247.

15. *GeForce RTX 30 Series* [online] [cit. 2020-10-20]. Dostupné z : https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/.

16. M., Clark. *CUDA Pro Tip: Kepler Texture Objects Improve Performance and Flexibility* [online] [cit. 2020-11-06]. Dostupné z : https://developer.nvidia.com/blog/cuda-pro-tip-kepler-texture-objects-improve-performance-and-flexibility/.

17. *CUDA Toolkit Documentation* [online] [cit. 2020-11-06]. Dostupné z : https://docs.nvidia.com/cuda/.

18. KHUDIAKOV, G. I. Sampling theorem in signal theory and its originators. *Journal of Communications Technology and Electronics*. 2008, roč. 53. Dostupné z DOI: 10.1134/S1064226908090118.

19. HADDAD, R. A.; AKANSU, A. N. A class of fast Gaussian binomial filters for speech and image processing. *IEEE Transactions on Signal Processing*. 1991, roč. 39, č. 3, s. 723–727. Dostupné z DOI: 10.1109/78.80892.

20. KERCKHOVE, Michael. *Scale-Space and Morphology in Computer Vision*. Springer, 2001. Lecture Notes in Computer Science. ISBN 9783540423171. Dostupné z DOI: 10.1007/3-540-47778-0.

21. RENNICH, Steve. *CUDA C/C++ Streams and Concurrency* [online]. NVIDIA [cit. 2020-11-16]. Dostupné z : https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf.

22. HOLUB, Petr; MATELA, Jiří; PULEC, Martin; ŠROM, Martin. UltraGrid: Low-Latency High-Quality Video Transmissions on Commodity Hardware. In: *Proceedings of the 20th ACM International Conference on Multimedia*. Nara, Japan: Association for Computing Machinery, 2012, s. 1457–1460. MM '12. ISBN 9781450310895. Dostupné z DOI: 10.1145/2393347.2396519.

23. MARK SEGAL, Kurt Akeley. *The OpenGL Graphics System:A Specification(Version 4.6 (Core Profile) - October 22, 2019)* [online]. The Khronos Group Inc. [cit. 2020-12-05]. Dostupné z : https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf.

24. *SDL_GL_CreateContext* [online] [cit. 2020-12-09]. Dostupné z : http://wiki.libsdl.org/SDL_GL_CreateContext.

25. *Load OpenGL Functions* [online] [cit. 2020-12-09]. Dostupné z : https://www.khronos.org/opengl/wiki/Load_OpenGL_Functions.

26. *Oculus SDK for Linux* [online] [cit. 2020-12-13]. Dostupné z : https://developer.oculus.com/downloads/package/oculus-sdk-for-linux/.

27. *API Documentation* [online] [cit. 2020-12-13]. Dostupné z : https://github.com/ValveSoftware/openvr/wiki/API-Documentation.

28. *SteamVR for Enterprise / Government Use* [online] [cit. 2020-12-13]. Dostupné z : https://partner.steamgames.com/doc/features/steamvr/enterprise.

29. *Devices* [online] [cit. 2020-12-13]. Dostupné z : http://www.openhmd.net/index.php/devices/.

30. *OpenXR Overview* [online] [cit. 2020-12-14]. Dostupné z : https://www.khronos.org/openxr/.

31. *VIVE OpenXR* [online] [cit. 2020-12-14]. Dostupné z : https://developer.vive.com/resources/openxr/.

32. *OpenXR overview* [online] [cit. 2020-12-14]. Dostupné z : https://docs.microsoft.com/en-us/windows/mixed-reality/develop/native/openxr.

33. *OpenXR Support for PC Development* [online] [cit. 2020-12-14]. Dostupné z : https://developer.oculus.com/documentation/native/pc/dg-openxr.

34. HEANEY, David. *Facebook Follows Valve & Microsoft In Recommending Game Engines Use OpenXR* [online] [cit. 2020-12-14]. Dostupné z : https://uploadvr.com/facebook-now-recommends-openxr/.

35. *OpenXR Support for PC Development* [online] [cit. 2020-12-14]. Dostupné z : https://monado.freedesktop.org/.

36. *The OpenXR Specification* [online] [cit. 2020-12-15]. Dostupné z : https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html.

37. *The Perspective and Orthographic Projection Matrix* [online] [cit. 2020-12-19]. Dostupné z : https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.

38. *Pixel Buffer Object* [online] [cit. 2020-12-06]. Dostupné z : https://www.khronos.org/opengl/wiki/Pixel_Buffer_Object.

39. *SDL_GL_SetAttribute* [online] [cit. 2020-12-20]. Dostupné z : https://wiki.libsdl.org/SDL_GL_SetAttribute.

40. *Vertex Specification* [online] [cit. 2020-12-20]. Dostupné z : https://www.khronos.org/opengl/wiki/Vertex_Specification.

41. *std::chrono::steady_clock* [online] [cit. 2020-12-21]. Dostupné z : https://en.cppreference.com/w/cpp/chrono/steady_clock.

42. *GEFORCE RTX 2070* [online] [cit. 2020-12-28]. Dostupné z : https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2070/.

43. *GEFORCE GTX 1080 Ti* [online] [cit. 2020-12-28]. Dostupné z : https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080-ti/.

44. *Bringing pixels front and center in VR video* [online] [cit. 2020-12-29]. Dostupné z : `https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/`.

A Elektronická príloha

Súčasťou práce je aj elektronická príloha dostupná v elektronickom archíve práce. Tento archív obsahuje zdrojový kód programu UltraGrid a knižnice na skladanie obrazu.

B Meranie výkonu a oneskorenia implementácie využívajúcej knižnicu VRWorks

Tento dodatok obsahuje porovnanie výkonu a oneskorenia výslednej implementácie skladania obrazu so staršou verziou, ktorá na skladanie využívala knižnicu VRWorks 360 Video vo verzii 2.1. Meranie bolo vykonané rovnakým spôsobom, ako bolo opísané v kapitole 8, na počítači **HD13**. Vstupné aj výstupné video je v rozlíšení 8K pri použití formátu obrazových bodov UYVY. Pri skladaní pomocou knižnice VRWorks bol využitý jej pipeline `NVSTITCH_STITCHER_PIPELINE_MONO_EQ`. Pre porovnanie sú vo výsledkoch obsiahnuté aj namerané výsledky výslednej implementácie zo sekcie 8.3.1. Namerané výsledky sú v tabuľke B.1.

Tabuľka B.1: Namerané hodnoty oneskorenia a výkonu.

Knižnica	Počet vstupných snímok za sekundu	Oneskorenie priemer [ms]	Smerodajná odchýlka oneskorenia [ms]	Priemerná perióda snímok [ms]	Smerodajná odchýlka periódy [ms]
gpustitch	60	29.31	0.06	17.01 (58.79 fps)	0.04
gpustitch	30	23.80	0.04	33.33 (30.00 fps)	0.05
vrworks	60	30.50	0.05	30.53 (32.75 fps)	0.06
vrworks	30	30.06	0.04	33.33 (30.00 fps)	0.05

Z nameraných hodnôt je vidieť, že verzia, ktorá používa knižnicu VRWorks dosahuje nižší výkon a väčšie oneskorenie. Pri analýze behu pomocou nástroja nvprof bolo možné pozorovať, že tento rozdiel je spôsobený prevažne tým, že knižnica VRWorks nevykonáva výpočet súbežne s pamäťovými prenosmi vstupných snímok.